uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Informática
2018-2019

Trabajo Fin de Grado

"Separación de voz de muestras multifuente"

Francisco Javier Alonso Rubio

Tutor/es

Ángel García Crespo

Colmenarejo, 2019

Resumen: En los últimos años, el número de aplicaciones informáticas que procesa la voz humana, con diversos fines, ha aumentado sensiblemente. Cualquiera de esas aplicaciones, para su correcto funcionamiento, necesita disponer de la voz lo más intacta posible. En este proyecto se desarrolla un sistema utilizando redes neuronales, con el propósito de extraer la voz humana de cualquier audio, eliminado en el proceso los demás sonidos encontrados.

Palabras clave: Redes Neuronales; Audio; Separación Voz; Separación de Fuentes; Fuente Sonora; NMF, MFCC;

[EN] Voice separation from multisource audio

Abstract: In recent years, the number of computer applications which process the human voice has increased sensibly. Any of these applications need the cleanest voice possible for their proper functioning. In this project, a system is developed using neural networks with the purpose of extracting the human voice from any audio, erasing in the process any other sound found.

Keywords: Neural Networks; Audio; Voice Separation; Source Separation; Sound Source; NMF, MFCC.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1 Motivación del trabajo

La voz es uno de los mecanismos de comunicación más primitivos que tenemos a nuestra disposición. Desde nuestros orígenes, desde un punto de vista práctico, la hemos utilizado para comunicarnos con otros seres, ya sean humanos o animales.

Pero en pleno siglo XXI, gracias a los avances de la informática, ha aparecido una nueva posibilidad: comunicarnos con las máquinas mediante la voz. Las máquinas han pasado a convertirse en receptores, en intérpretes de la voz humana.

En los últimos quince años se han conseguido avances significativos en este novedoso campo. Se han desarrollado gran cantidad de aplicaciones de diversa índole que utilizan la voz humana como medio de entrada comunicativo:

Asistentes personales (Alexa, Siri), bots conversacionales (Mitsuku), transcriptores voz-texto (Google Transcription), traductores (Translate Voice), sistemas biométricos (VoicePIN), sistemas de vigilancia (ECHELON), videojuegos (Oculus Voice), entre otros.

Todas estas aplicaciones tienen en común la necesidad de que la voz entre al sistema en el mejor estado posible para funcionar de forma óptima. La voz no es el único habitante del espectro sonoro, sino que coexiste, simultáneamente, con una gran variedad de sonidos. Esto produce un enmascaramiento de la voz, es decir, dificultad de identificar a la voz. Bajo este contexto, han de buscarse soluciones que permitan recuperar la voz humana lo más intacta posible para que aplicaciones como las antes mencionadas puedan funcionar correctamente.

1.2 Objetivos

El objetivo principal es desarrollar un sistema que, introduciéndole audio como entrada, lo procese de tal manera que solo permanezca la voz humana, eliminando cualquier otra fuente o sonido que no sea identificado como voz.

Se contemplarán otros objetivos, secundarios, como los siguientes:

- El sistema funcionará en tiempo real, de tal manera que el audio procesado sea obtenido tras un leve retardo respecto al audio de entrada.

- El sistema funcionará de igual manera en el primer instante de audio como en el último. Es decir, el sistema tendrá la misma efectividad a lo largo de su funcionamiento continuado.

- El sistema será, preferiblemente, multiplataforma, capaz de ser ejecutado en sistemas operativos de escritorio como Windows o basados en Linux.

1.3 Metodología

La metodología seguida para realizar este trabajo consiste en la realización de las siguientes tareas, de forma no secuencial, puesto que en ocasiones se han encontrado dificultades o problemas no previstos que han requerido retornar a fases previas:

- **Investigación**: se ha realizado una búsqueda de soluciones posibles al problema en cuestión y se han identificado para cada una sus pros y sus contras. También se han tenido que estudiar una serie de conceptos necesarios referentes al campo del sonido y de procesamiento digital de señales. Gracias a un interés previo ya existente hacia ambos mundos, se disponía de cierto bagaje conceptual, así como una comprensión de muchos conceptos posteriormente utilizados en la solución escogida. Como proceso de investigación se considera también la búsqueda de datos (*datasets*) a utilizar.

- **Diseño**: durante esta fase se establece la arquitectura del sistema a implementar y las funcionalidades que este ofrece, con el fin de implementar el sistema de forma más eficiente y efectiva. Otra utilidad de realizar el diseño es la generación de diagramas facilitan la comprensión del sistema.

- **Implementación**: esta fase consiste en materializar el diseño previo en código. En concreto, código en lenguaje Python. Esta fase se estima haber llevado entorno al 80% del tiempo total del desarrollo del proyecto.

- **Pruebas**: esta fase es crítica y no debe confundirse con las pruebas asociadas a la correcta implementación y funcionamiento del código, sino con la generación de diferentes resultados al ejecutar el sistema con una serie de parámetros concretos, con el fin de que se cumpla el objetivo principal establecido previamente. Decenas de pruebas han sido ejecutadas, almacenadas y comparadas entre sí para poder determinar que valores paramétricos son los mejores.

- **Documentación**: fase relativa a la creación de este documento, así como a la recopilación sistemática de fuentes utilizadas y otra información relevante a lo largo de la realización del proyecto.

1.4 Estructura

La estructura elegida en este documento sigue un orden lógico de acuerdo a las fases explicadas en el apartado de Metodología. En primer lugar, se incide en la fase de investigación, con una explicación del estado del arte, así como de los conceptos teóricos en los que se basa la solución propuesta. A continuación, se indica la composición y preprocesamiento de los datos utilizados por el sistema. Posteriormente, se explica la implementación del sistema, apoyada de diagramas explicativos del mismo. En este apartado se discuten otras cuestiones técnicas como las librerías utilizadas o problemas relativos a la optimización del código. Después, se especifican las pruebas realizadas, sus resultados y un análisis comparativo de estos. Por último, se finaliza el documento con una serie de conclusiones extraídas, indicando si los objetivos establecidos se han cumplido y planteando posibles mejoras para el sistema.

1.5 Marco regulador

Para la realización de este proyecto se han de tener en consideración dos cuestiones. En primer lugar, qué tipo de licencias aplican tanto a los *dataset* como de las librerías o software utilizados. En segundo lugar, de acuerdo a dicho tipo de licencias, qué condiciones se aplicarán si el proyecto es puramente académico o si busca comercializare.

En este proyecto, para la implementación del código se utilizan únicamente librerías *open source* como *Tensorflow*, *libROSA*, *Ludwig* o *PyAudio*. *Tensorflow* emplea una licencia del tipo Apache 2.0, la cual permite su uso con fines comerciales, incluso para la creación de patentes. *libROSA* utiliza una licencia de tipo ISC, la cual es similar en efecto prácticos a una licencia de tipo MIT, por tanto, se permite utilizar para uso comercial. *Ludwig*, librería basada en *Tensorflow*, utiliza al igual que esta una licencia Apache 2.0. *PyAudio* utiliza una licencia MIT por su parte. Otras librerías utilizadas como *NumPy, SciPy* o *Pandas* utilizan una licencia de tipo BSD, permitiendo también la comercialización sin apenas restricciones (suele requerirse tan sólo la debida mención del copyright de la librería utilizada en el proyecto).

Por otro lado, más interesante aún son las cuestiones generadas por el uso que se le da a los *datasets.* Supongamos que utilizamos uno o varios *datasets* para entrenar una red

neuronal (como se hará en este proyecto) y posteriormente comercializamos el sistema. Los *datasets*, una vez utilizados para entrenar la red neuronal, no tienen por qué formar parte del sistema y ni siquiera podría quedar constancia de su utilización. ¿Cómo se puede asegurar la participación o uso de cierto *dataset* para la creación de un modelo, resultado de entrenar una red neuronal? Surgen también, por tanto, una serie de cuestiones éticas respecto al uso que se le da a los datos. Técnicamente, el acceso a un *dataset* suele pertenecer a una categoría, la cual puede ser más o menos restrictiva: datos privados, datos compartidos o datos abiertos.

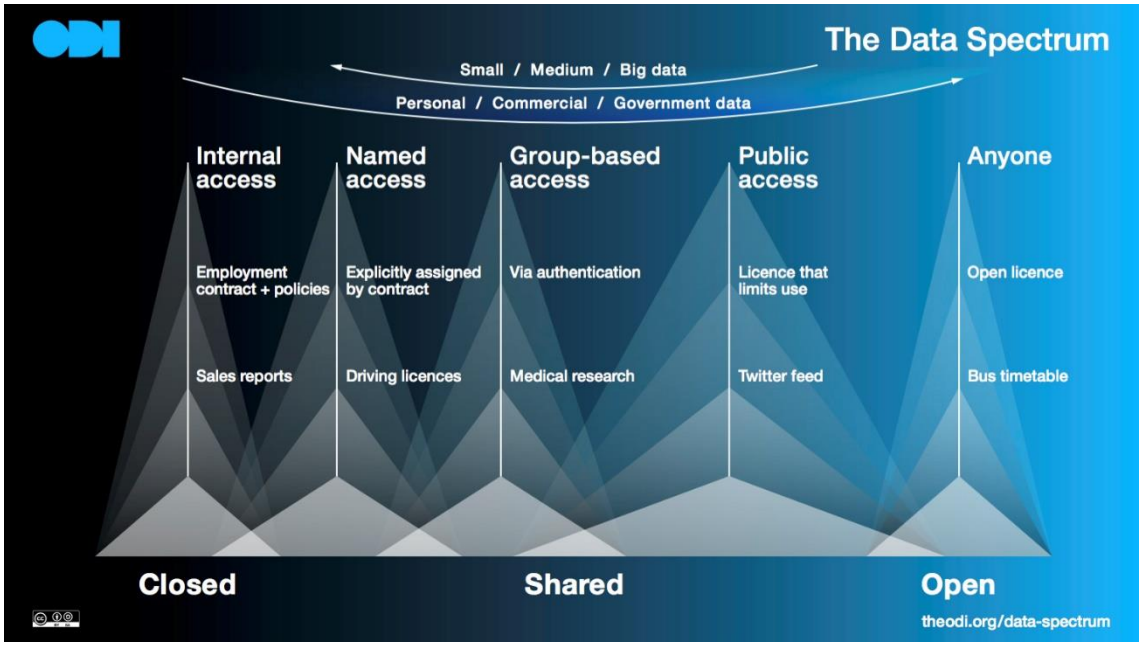

Figura 1: El espectro de datos[1]

Todos los *datasets* utilizados en este proyecto pertenecen a la categoría de datos abiertos, en concreto de acceso público. No se ha requerido proporcionar identificación o prueba de pertenecer a una institución académica (como ocurre en otros *datasets*). En el capítulo de Preparación de los *Datasets* se mencionarán los *datasets* utilizados en este proyecto.

Ha sido difícil interpretar el tipo de licencia que utiliza cada *dataset* ya que en la mayoría de las veces viene acompañado de algún tipo de software que sí posee una licencia concreta, pero no se menciona que ocurre por dar uso únicamente de los datos. Sin embargo, ya que no se pretende dar un uso comercial a este proyecto (en el capítulo de

[1] https://theodi.org/about-the-odi/the-data-spectrum/

Conclusiones se determinará por qué), se han citado los *datasets* utilizados con un fin únicamente académico.

1.6 Entorno socio-económico

Este proyecto, en el caso de desarrollarse para alcanzar un estado de potencial comercialización, podría realizarse en su mayoría mediante contenido gratuito, *open source*, como se ha especificado en el apartado anterior. Por otro lado, como se indicará en el capítulo de Estado del Arte, no existen muchos productos en el mercado de este tipo. Por tanto, la competición, en caso de comercializar el sistema, se produciría con muy pocos productos, lo cual facilitaría el posicionamiento del sistema en el mercado. Además, este sistema se especializaría únicamente en la tarea de extraer la voz, donde la gran mayoría del resto de productos se centran en otra serie de funcionalidades.

Este sistema encaja mejor como un añadido a otros productos, por ello, podría plantearse la idea de desarrollarse como si de un "plugin" se tratase. Por ejemplo, comercializarlo a modo de librería, para que otros desarrolladores puedan adquirirla e incorporarla en el funcionamiento de sus productos.

Puesto que, como se ha indicado en el párrafo anterior, este sistema tiene sentido como parte de otro sistema o producto principal, es difícil medir el impacto socio-económico que podría tener ya que dicho impacto dependería del producto principal al que se integraría. En el apartado de Motivación del Trabajo se ha mencionado una serie de posibles aplicaciones en las que se integraría este sistema. Dichas aplicaciones pueden ser de uso comercial, personal o incluso gubernamental. Esto es otro factor que dificultaría el impacto causado por el sistema, al ser potencialmente aplicable en aplicaciones de diversa índole y sectores muy diferentes.

2. ESTADO DEL ARTE

2.1 Soluciones existentes

El problema al que nos enfrentamos es extraer la voz de muestras o audios multi fuente. Una fuente, en este contexto, puede definirse como el origen de un conjunto de señales de audio. Es decir, una fuente podría ser una persona, un violín, una guitarra, un coche, un perro, etc. Cada una de estas fuentes genera conjuntos de señales auditivas diferentes

entre sí (aunque existe la posibilidad de imitaciones o sonidos parecidos). En este proyecto nos centramos en las fuentes de habla humana, considerando toda otra fuente como si de ruido se tratase.

En el ámbito del procesamiento de audio es muy popular el concepto de Separación Ciega de Fuentes (Blind Source Separation). Consiste en la separación de fuentes sin apenas información previa sobre la naturaleza de las fuentes existentes en la muestra. Para conseguir la separación se suele utilizar una técnica específica como PCA [1], ICA [1] o NMF [2].

Sin embargo, es difícil identificar cual es el Estado del Arte. No existe un consenso o una solución reconocida como la mejor. Parte de este problema se debe a que la mayoría de las aplicaciones de este campo no son de código abierto y el conocimiento, por tanto, es limitado. Algunas aplicaciones del tipo Software Propietario son *Audionamix XTRAX STEMS 2*, *iZotope RX 7* o *AudioSourceRE*. El primer software de código abierto de separación de fuentes fue *openBlissart*, que fue lanzado en 2011. En 2016 se lanzó la librería *untwist* y esta proporciona diferentes métodos para la separación de fuentes. Una de las librerías más recientes es *Nussl* (2018). Sin embargo, no existe una solución de código abierto considerada Estado del Arte [9].

Otro factor influyente en este problema es que gran parte de las soluciones propuestas en los últimos años utilizan técnicas basadas en *Deep Learning* y los datos necesarios para sustentar estos sistemas requieren ser muy numerosos. Obtener muestras de audio diversas y de calidad es un proceso costoso además de difícil de automatizar. Sin embargo, actualmente ya se dispone de acceso público a *datasets* de gran tamaño y de diversa índole.

Por otro lado, de acuerdo con los objetivos establecidos, hay que considerar una serie de limitaciones a la hora de escoger la técnica a utilizar. En primer lugar, la muestra o audio de la que extraer la voz debe ser de un único canal, es decir, mono. Se han desarrollado técnicas como DUET [6] que utiliza las diferencias existentes entre los canales estéreo para realizar la separación, pero en este proyecto no podrá utilizarse al procesarse muestras de forma mono. Otro factor limitante es que nuestro sistema necesita funcionar sin necesitar información previa. Es decir, no se le debe especificar información sobre la muestra a procesar como, por ejemplo, secciones donde únicamente existe ruido y secciones donde se encuentre la fuente a extraer. La técnica REPET [7], especializada en

separar la voz de la música, utiliza a su favor el carácter repetitivo de esta última, pero, pese a ser tan eficaz como NMF en esta tarea [8], no es posible usar esta técnica al no ser tan efectiva ante ruidos aleatorios o sonidos carentes de estructura repetitiva; además necesita de información previa para funcionar correctamente al igual que DUET.

Se ha decidido utilizar NMF principalmente, por tres motivos:

1) Popularmente utilizada estos últimos años.

2) Ha demostrado ser efectiva en diferentes situaciones [2] [4] [5].

3) Más intuitiva que otras técnicas existentes.

En el siguiente apartado se explica en qué consiste NMF y qué uso se le da en este proyecto.

2.2 Factorización no Negativa de Matrices (NMF)

Dada una matriz A, factorizar dicha matriz consiste en descomponerla como el producto de dos o más matrices:

$$A \approx BC$$

Técnicas conocidas para la factorización de matrices son SVD, QR o NMF. Esta última tiene la peculiaridad de ser una factorización donde todos los elementos son no negativos. Particularmente, en NMF la notación suele ser la siguiente:

$$V \approx WH$$

V es la matriz de datos originales no negativos. Cada columna es una muestra de los datos y cada fila corresponde a una característica de los datos. El espectrograma del audio a descomponer se utilizará como datos de *V*. Utilizar la magnitud del espectrograma es ideal ya que se compone de valores no negativos, tal y como requiere NMF. En el siguiente capítulo se entrará en detalle en qué consiste el espectro de un audio (información mostrada en un espectrograma).

W es la matriz de *k* vectores base, también conocida como elementos del diccionario. Cada columna es un vector base.

Por último, *H* es la matriz de pesos o ganancias. Cada fila representa la ganancia del vector base correspondiente.

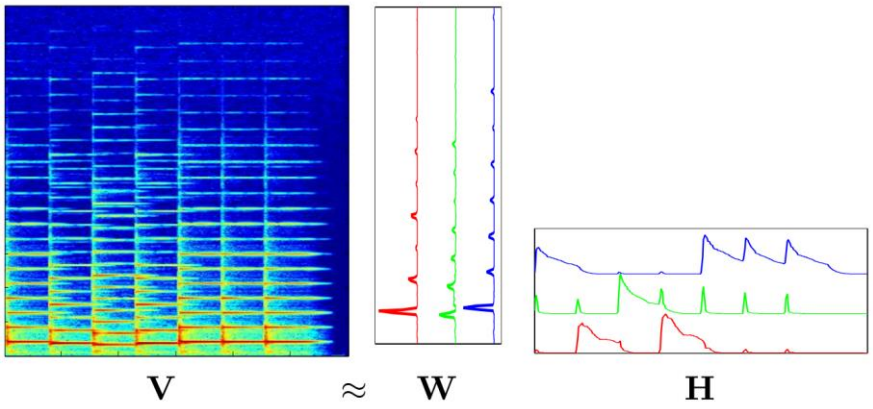

Figura 2: Descomposición NMF de un espectrograma V con k=3 componentes [3]

Los componentes obtenidos tras la descomposición NMF, sumados entre sí, generarán como resultado una aproximación de la matriz *V* original. Cuanto mayor sea el valor de *k*, es decir, el número de componentes, mayor será la correlación o semejanza con los datos originales. Realizando algunas pruebas programáticas, se han obtenidos valores en torno al 99% de correlación de Pearson con $k = 16$ componentes para diferentes audios. Con valores de *k* muy pequeños, menores de 10, este porcentaje se decrementa sensiblemente y en la recomposición del audio original se llegan a apreciar artefactos. Se ha comprobado, además, que cuanto más complejo es el espectrograma original *V*, más componentes se necesitan para reconstruirlo correctamente.

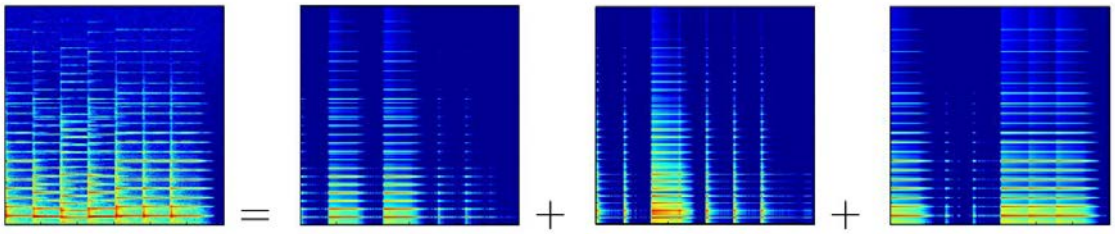

Figura 3: aproximación de V como la suma de 3 componentes [3]

Aún no se ha explicado cómo se obtienen las matrices *W* y *H*. Se trata de un problema que busca minimizar la divergencia entre la matriz de datos original *V* y la matriz recompuesta *V'*, la cual aparece en la Figura 2 como la suma de sus componentes. En concreto, se utilizará el algoritmo Kullback- Leibler para minimizar dicha divergencia. Bajo esta forma, NMF toma el nombre NMF-KL.

No se entrará en detalle en explicar dicho algoritmo puesto que nos alejaríamos del objetivo principal de este documento. NMF-KL es una herramienta más que utilizamos para intentar alcanzar nuestro objetivo y su implementación ya es dominio público desde hace años.

3. SOLUCIÓN PROPUESTA

En este punto sabemos que NMF es útil para descomponer el espectrograma de una señal de audio en *k* componentes. De esta manera, podríamos teorizar que, sumando ciertos componentes, se podrá obtener una fuente concreta, sumando otros componentes se obtendrá otra fuente y así sucesivamente, hasta recuperar los distintos componentes presentes en una señal de audio.

La elección definitiva de NMF no solo viene motivada por la literatura previa, sino gracias al resultado experimental. Se realizó la siguiente prueba:

Tomando dos señales de audio diferentes, claramente identificables por separado, sea un bombo de batería y un piano, se suman en una única señal, la mezcla. De dicha mezcla se trata de separar sus elementos, mediante NMF. Se obtienen *k* componentes (en este caso se usaron 16), los cuales, al ser espectrogramas, se convierten en señales de audio. Esto nos permite evaluarlos con el oído, buscando discriminar los componentes que pertenecen al bombo y los que pertenecen al piano. Se observa que una manera más sencilla de realizar esta separación "manual" es elegir aquellos componentes reconocibles como parte de sólo una de las fuentes, por ejemplo, escoger aquellos componentes que el oído reconozca como bombo. Este es el origen de la idea de tratar los audios como mezclas de solo dos fuentes, voz, de la cual recogeremos información sobre su naturaleza, y ruido, que será todo aquello no reconocido como voz. Esta heurística es un pilar fundamental en el proyecto.

Volviendo a la tarea anterior, se comprueba que la separación se ha realizado de forma efectiva, comparando las fuentes con sus respectivas muestras originales. Sin embargo, pese a que bajo estas condiciones se logra la separación, ante el problema general surgen dos cuestiones:

1) ¿Cómo desarrollar un sistema que sea capaz de elegir los componentes de cada fuente de forma automatizada, sin requerir de la intervención y criterio humanos?

2) ¿Es capaz de realizarse la separación en mezclas más complejas, con más de dos fuentes simultáneas en las que ocurran solapamientos de partes de sus espectros?

Se utilizará una red neuronal para intentar solucionar la primera cuestión, la cual es esencial para la consecución del objetivo principal. La segunda cuestión está ligada a la primera, puesto que la capacidad de separar señales complejas será consecuencia de la correcta implementación del sistema. La red neuronal funcionará como una red neuronal clasificadora de componentes NMF en dos categorías, voz y ruido.

El siguiente paso es definir qué tipo de datos manejará la red neuronal clasificadora para su funcionamiento. Los datos consistirán en características representativas de los componentes, de los que, en la fase de entrenamiento, sabemos de antemano si proceden de voz o de ruido. Estas características son características espectrales, es decir, características cuya información es extraída a partir del espectro y es, por tanto, información no temporal, instantánea, sobre las frecuencias presentes (también llamados armónicos o parciales) y la magnitud de estas. Para obtener el espectro del audio de un componente se ha realizar la Transformada de Fourier de dicho audio. Técnicamente, a nivel de implementación, se utiliza la llamada Transformada Rápida de Fourier o FFT.

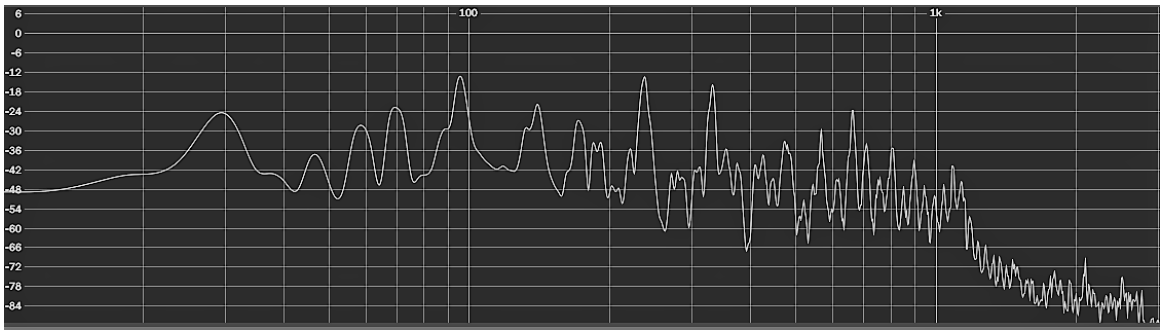

Figura 4: espectro de una onda de audio en un instante dado

El espectro, como representación de una onda de audio, nos permite analizar dicha onda, así como de poder categorizar a la misma. La estructura armónica, lo que llamamos timbre, contiene información sobre la naturaleza del sonido. Sonidos diferentes tendrán espectros diferentes. Opuestamente, sonidos similares tendrán espectros similares.

En la voz se pueden encontrar formas reconocibles, patrones, en su espectro. Estos son los **formantes**, formas espectrales que se corresponden a las vocales humanas. Analizando el espectro de un sonido podría estimarse si la estructura se asemeja a la de

una vocal concreta y, por tanto, determinar que el sonido procede de una voz humana. Esta idea es clave en la elección de la solución. Si en la voz humana se pueden encontrar patrones, en gran medida generalizables, existe la posibilidad de que nuestra red neuronal clasificadora sea capaz de detectar por si misma estos patrones, además de otros no contemplados. En contrapartida, las consonantes son un problema, ya que carecen de una estructura tan definida como la de las vocales. Este problema se encontrará en posteriores secciones.

Teniendo ya una ligera concepción de qué es un espectro, las características espectrales que utilizará la red neuronal son las siguientes:

1) **Magnitud Media de Conjuntos de Frecuencias (MMCF)**. Al principio, se consideró utilizar como característica cada una de las magnitudes de las frecuencias determinadas por la FFT, pero es mucho más eficiente agruparlas en *c* conjuntos y así comprimir la información. Si se realiza la FFT de una señal con un tamaño de ventana de *n* muestras, se tendrá una resolución, en el espectro, de n+1/2, frecuencias diferentes. Por ejemplo, la FFT de un valor de ventana de 2048 muestras (*samples)* para una frecuencia de muestreo de 16000 Hz, generará un espectro capaz de representar la onda transformada desde la frecuencia 0 Hz a 8000 Hz (teorema de Nyquist) con 1025 frecuencias distintas entre ambos límites, conteniendo los extremos. A cada una de estas frecuencias se les suele denominar armónicos (la terminología correcta sería "parciales" puesto que no todo parcial es armónico, pero se utilizará el primer término para facilitar la comprensión). La agrupación consistirá entonces en comprimir esas 1025 magnitudes en un número sensiblemente menor, como puede ser 64, realizando la media entre grupos (de igual tamaño; 16 magnitudes por grupo si descartamos la primera magnitud, el DC Offset) de magnitudes contiguas. En resumen, con esta característica obtenemos una representación básica de la forma del espectro en sus diferentes "bandas" o secciones. De ahora en adelante nos referiremos a esta característica, por decisión propia, como MMCF.

2) **Coeficientes Cepstrales en las Frecuencias de Mel (MFCC)**. Es una característica espectral ampliamente utilizada en aplicaciones de voz. Se ha probado que es una característica efectiva en tareas de reconocimiento de voz entre otras y es considera Estado del Arte desde los años 80, momento en el que

se introdujo. Esta característica ajusta las magnitudes del espectro de acuerdo a la percepción del oído humano, generando una serie de valores (este número se indica previamente) que representan a dicho espectro. Se utilizarán 12 valores MFCC en todas las pruebas.

3) **Llanura espectral (Spectral flatness)**. Esta característica representa en qué medida el espectro es llano. Esto nos sirve para identificar si un espectro es ruidoso o poco armónico, carente de estructura armónica. Se mide por un valor de 0 a 1, donde 1 indica llanura total del espectro. [5]

4) **Centroide espectral (Spectral centroid)**. La última característica incorporada es la frecuencia centroide del espectro. Esta característica puede ser útil para intentar identificar el "tono" del espectro. En otras palabras, nos indica alrededor de qué frecuencia reside la mayor cantidad de energía [5].

Ya se han definido el tipo de datos que utilizará la red neuronal, pero antes han de generarse los datos con los que entrenarla. Para ello se han recopilado una serie de *datasets* existentes, de acceso libre.

En definitiva, la solución consistirá en lo siguiente: se realizará la descomposición NMF de todas las muestras de dichos *datasets* y se obtendrán las características espectrales de cada uno de los componentes. Los valores de estas características se almacenarán y de esta manera se generará un *dataset* propio con el que entrenar la red neuronal clasificadora. Una vez entrenada la red, se podrá utilizar para separar la voz en un audio de entrada, en tiempo real.

Como se puede observar, se diferencian dos modos de funcionamiento del sistema: un modo de **Generación de datos y entrenamiento** y un modo de **Predicción**.

4. PREPARACIÓN DE LOS DATASETS

4.1 Introducción

En este capítulo se abordará la tarea de recopilar un conjunto de muestras de audio, de voz y, por otro lado, de ruido, con las que generar los datos de entrenamiento de nuestra red neuronal. Esto es, obtener una serie de *dataset* de ambas categorías.

Se aplicará, además, un procesamiento previo a todas las muestras para procurar que sean lo más homogéneas posible puesto que tendrán diferentes duraciones y amplitudes máximas al proceder de distintos *datasets*.

4.2 Elección de *datasets*

Para las muestras de voz, se han recopilado *datasets* formados por frases o palabras leídas por diferentes personas, con voces de diferente timbre y dicción. Los *datasets* utilizados presentan un equilibrio entre muestras de hombres y muestras de mujeres. Típicamente corresponden a edades adultas del rango de 20 a 50 años.

Para las muestras de ruido, se ha creado una categorización simple entre muestras de Ruido Urbano y muestras de Instrumentos. Las muestras de la primera categoría consisten en impactos, golpes, animales, vehículos, en definitiva, ruidos, en el sentido común de la palabra. La segunda categoría se compone de instrumentos como violines, baterías (tanto reales como sintetizadas), pianos, sintetizadores, guitarras, entre otros.

En la siguiente tabla se recogen los *datasets* utilizados:

Nombre	Categoría	Subcategoría	Número de muestras utilizadas
Warden P. Speech Commands [D1]	Voz	-	3500
CMU PDA Database [D2]	Voz	-	835
TED Dataset	Voz	-	5500
VoxForge SpeechCorpus [D3]	Voz	-	14000
IRMAS [D4]	Ruido	Instrumento	6000
NSynth Dataset [D5]	Ruido	Instrumento	12275
Drum-machines [D6]	Ruido	Instrumento	7476
UrbanSound8K Dataset [D7]	Ruido	Ruido Urbano	8732
Acoustic Event Dataset [D8]	Ruido	Ruido Urbano	5223

TABLA 1: *DATASETS* UTILIZADOS

TED *Dataset* ha sido creado de forma propia a partir de 260 charlas TED disponibles gratuitamente en *YouTube*. Se han extraído los audios de los videos y se han eliminado 25 segundos, tanto del principio como del final del audio, para evitar incluir música o aplausos que típicamente aparecen en ambas secciones.

4.3 Preprocesamiento de los *datasets*

En cuanto al preprocesamiento de todas las muestras, se ha realizado mediante la aplicación de edición de sonido *WavePad*.

WavePad permite el procesamiento de muestras por lotes (*batch processing*), por tanto, es un programa adecuado para procesar gran número de muestras. En la figura siguiente se muestra el procesado que sufrían las muestras de todos los *datasets*:

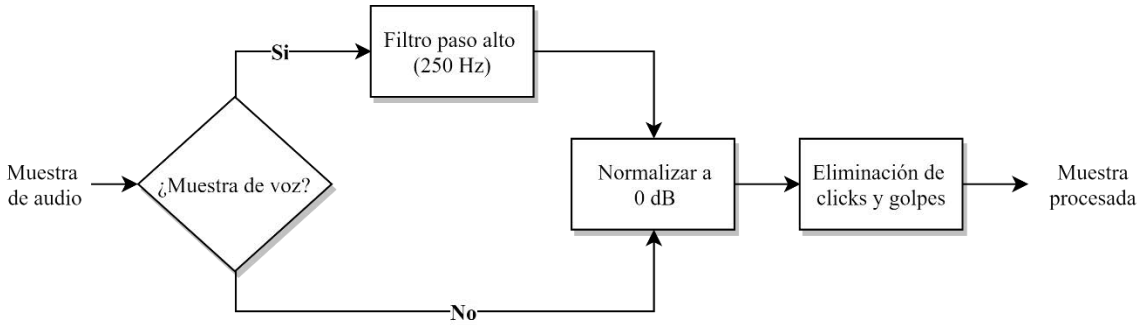

Figura 5: preprocesamiento de las muestras

Como se puede observar en la figura, se aplica un filtro paso alto a las muestras de voz para minimizar la cantidad de información existente en el rango 0 – 250 Hz, ya que es una banda poco relevante para la voz. En cambio, existen ruidos o instrumentos que poseen cantidades significativas de información en dicha banda, por lo que no se aplica el filtro en ese caso.

A continuación, se normaliza la muestra a 0 dB para procurar que los picos máximos de todas las muestras se encuentren al mismo nivel. Se ha decido no aplicar compresión, principalmente para dejar las muestras lo más intactas posibles. Finalmente se han eliminado los *clicks* que pudiesen encontrarse en las muestras (típicamente al principio o al final).

Las muestras se han agrupado en dos carpetas diferentes: VOICE y NOISE. Esto es por cuestiones de diseño, ya que el sistema necesitará conocer la ruta de la carpeta donde se encuentran las muestras voz y la ruta de la carpeta de las muestras de ruido.

En el siguiente capítulo, Implementación del Sistema, se especifica cómo se utilizan las muestras de estos *datasets*, ya listos para usar.

5. IMPLEMENTACIÓN DEL SISTEMA

5.1 Introducción

Como se ha mencionado en el apartado de Solución Propuesta, existen dos modos de funcionamiento diferentes:

1) **Generación de datos y entrenamiento**

2) **Predicción**

Es necesario indicar que la funcionalidad de **Predicción** únicamente podrá ejecutarse tras haber ejecutado el primer modo de funcionamiento (Generación de datos y entrenamiento).

Se ha optado por explicar el sistema mediante diagramas de flujo. Cada figura rectangular de los diagramas corresponde a una función real, implementada en el código. Se han obviado ciertas funciones, no esenciales, con el fin de no dificultar el objetivo de estos diagramas, que es explicar el funcionamiento esencial del sistema.

A continuación, se indicarán las principales librerías usadas para implementar el código. Posteriormente se explicarán ambos modos de funcionamiento del sistema. Por último, se abordará la cuestión de la Optimización de dicho sistema, vital en este proyecto.

Se puede acceder al código de este proyecto (repositorio *Github*) en el siguiente enlace[2].

5.2 Librerías utilizadas

Como se mencionó en el apartado de Marco Regulador, se han utilizado diversas librerías como *LibROSA*, *SciPy*, *PyAudio*, *Pandas* o *NumPy*. Algunas de estas han proporcionado implementaciones de algunas necesidades básicas del sistema como son la descomposición NMF, la obtención de los MFCC, la FFT o la obtención de las características "*flatness*" y "*centroid*". Para la implementación de la red se ha utilizado la librería *Ludwig*, la cual está basada en la conocida librería *Tensorflow*. *Ludwig* es capaz de crear una red neuronal a partir de la definición de un modelo de datos, es decir, nombre y tipo de datos de entrada y de salida de la red neuronal, y posteriormente entrenarla y usarla para predecir.

[2] https://github.com/javinsky/nmf_voice_extraction

5.3 Generación de datos y entrenamiento

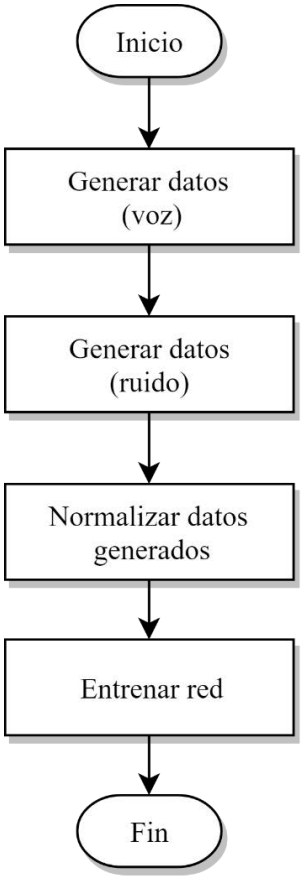

Figura 6: diagrama de secuencia del modo de funcionamiento 1 del sistema

El modo de funcionamiento 1 se compone de 4 fases claras. En primer lugar, se generan los datos para entrenar la red neuronal, a partir de las muestras de voz. Estos datos se almacenan un fichero *csv*. Posteriormente, se realiza el mismo proceso, pero esta vez con las muestras de ruido. El tercer paso es el de normalización de los datos generados. Por último, se realiza el entrenamiento de la red neuronal con los datos normalizados.

Teniendo este diagrama como referencia, se explicará en detalle cada una de sus 4 fases.

5.3.1 Generar datos (voz)

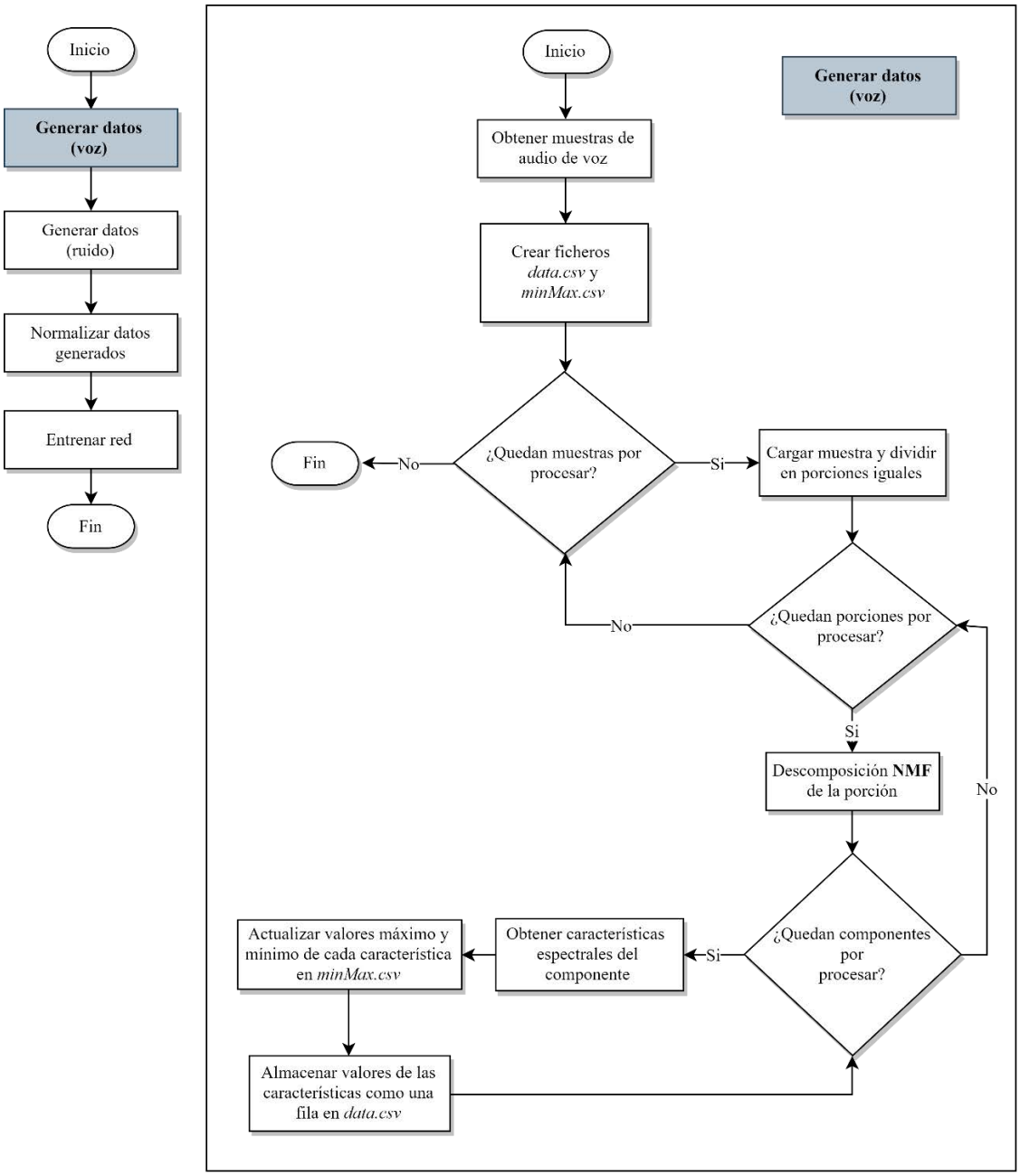

Figura 7: diagrama de secuencia de la fase de Generar datos (voz)

En esta primera fase se generan los datos de voz necesarios para entrenar la red.

Se comienza por obtener las muestras de voz, las cuales estarán almacenadas en una carpeta definida previamente. A continuación, se crea el fichero *data.csv* donde se almacenarán los datos generados a partir de dichas muestras de audio. Las columnas del fichero *data.csv* corresponden a las características espectrales, especificadas anteriormente en Solución Propuesta, de la siguiente forma:

Típicamente, MMCF comprenderá 64 columnas (*mmcf_01*, *mmcf_02*, etc.), y MFCC otras 12 columnas (*mfcc_01*, *mfcc_02*, etc.). Por otro lado, *S.Flatness* y *S.Centroid* sólo necesitarán una columna cada una (*flatness* y *centroid*). Por último, existirá una columna llamada *voice* que tendrá un valor binario de 0 o 1. Tomará el valor 0 si la fila proviene de una muestra de ruido y tomará el valor 1 si procede de una muestra de voz. En esta fase, todas las filas introducidas tendrán, por tanto, el valor 1 en esta columna. El número de columnas MMCF variará según el tamaño de ventana usado para calcular la Transformada de Fourier (FFT). A mayor tamaño de ventana, mayor número de columnas y viceversa.

También se creará un fichero llamado *minMax.csv* donde se almacenará el valor máximo y mínimo de cada característica (columna) de *data.csv*. Calcular el máximo y el mínimo de cada columna de un fichero *csv* es un proceso muy costoso cuando el fichero es muy grande (de varios gigabytes). Por ello, cada vez que se introduce una nueva fila en *data.csv,* se actualizan los valores de *minMax.csv,* con lo que ya no es necesario realizar la lectura de cada columna.

En este punto se comienzan a procesar todas las muestras. El procesamiento de una muestra consiste, en primer lugar, en separar dicha muestra en pequeñas porciones del mismo tamaño. Posteriormente, para cada porción se realiza su descomposición NMF en *k* componentes. De cada componente se obtienen sus características espectrales (MMCF, MFCC, *S.Flatness* y *S.Centroid*) y se almacenan como una fila en el fichero *data.csv*.

Para comprender la magnitud de los datos generados supongamos el caso de procesar una muestra de 1 segundo de duración habiendo establecido un tamaño de porción de 1024 *samples*. Como en el proyecto se trabaja a una frecuencia de muestreo de 16000 Hz, dicha muestra contendrá 16000 *samples*, que se puede dividir en 15 porciones de 1024 *samples* (se descartan los *samples* sobrantes). Si a cada porción se realiza la descomposición NMF en $k = 16$ componentes (este será el valor mínimo, en otras pruebas se usarán valores de hasta 96 componentes) esto da como resultado un total de 240 filas de datos a insertar. En conclusión, los datos generados tienen un tamaño mayor que el de las muestras de audio de las que se extraen los datos.

5.3.2 Generar datos (ruido)

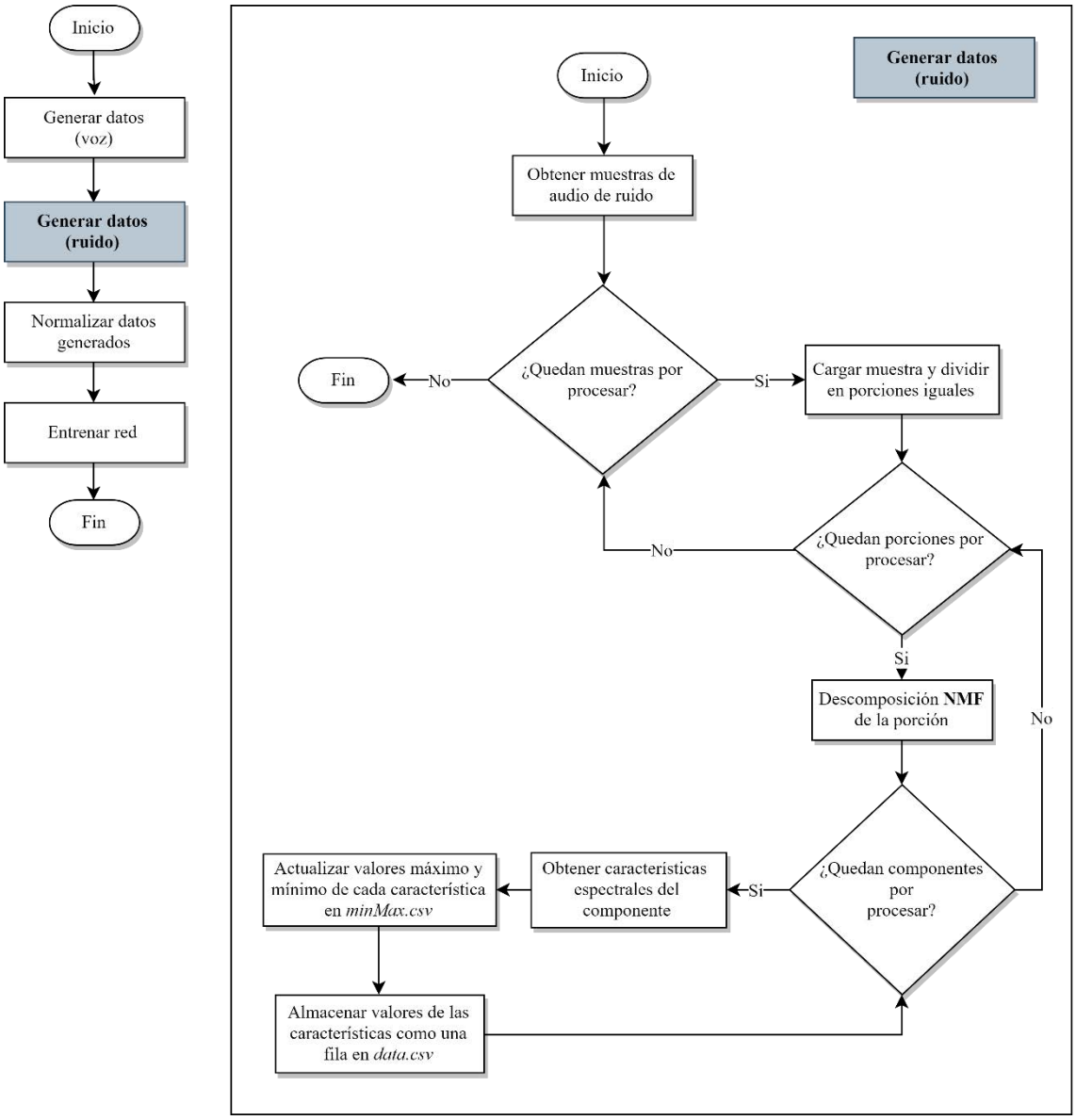

Figura 8: diagrama de secuencia de la fase de Generar datos (ruido)

Esta fase es prácticamente igual que la anterior con la diferencia de que se utilizarán muestras, esta vez, de ruido. Tampoco se necesita crear los ficheros *data.csv* y *minMax.csv* puesto que ya fueron creados en la fase anterior.

Como ya se ha mencionado anteriormente, en este caso, en la columna *voice* de los datos generados se almacenará el valor 0 para todas las filas nuevas, indicando que los datos generados en esta fase procederán de muestras de ruido.

Finalizada esta fase, se realiza la normalización los datos de *data.csv* tal y como se especifica en el apartado siguiente.

5.3.3 Normalizar datos generados

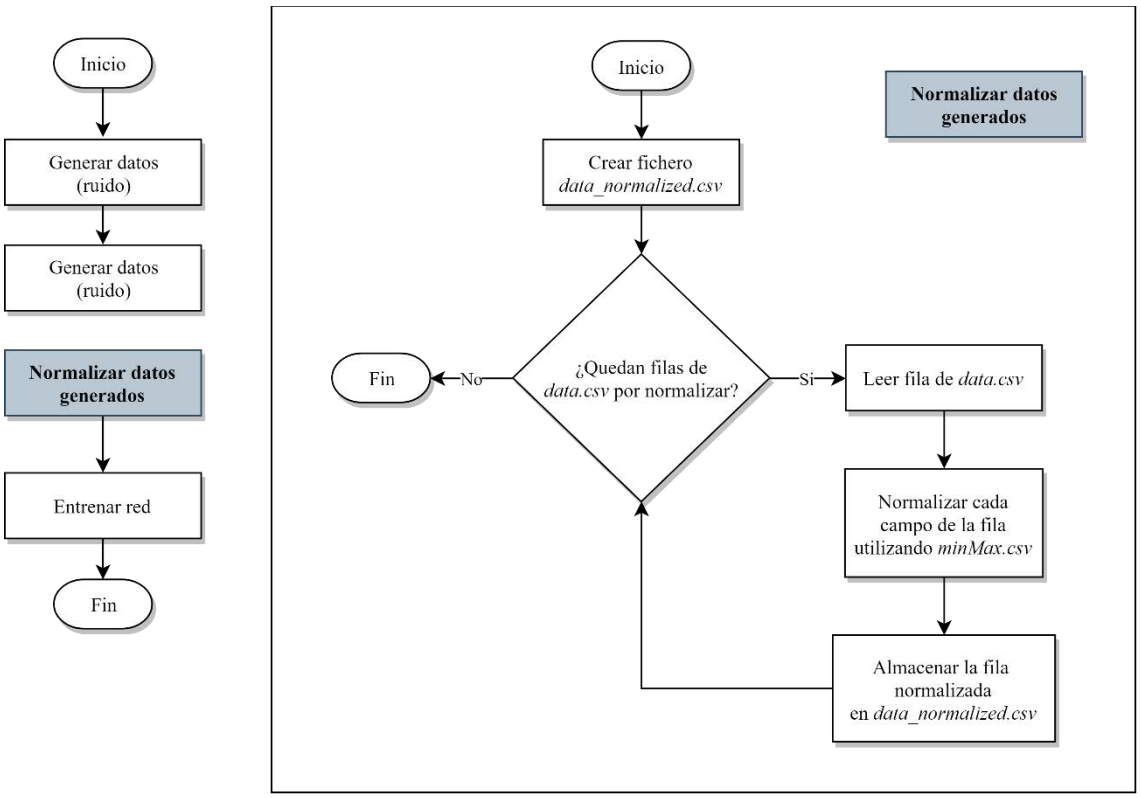

Figura 9: diagrama de secuencia de la fase de normalización

Puesto que los datos generados contienen características cuyos valores oscilan entre valores de diferentes rangos es recomendable normalizarlos de cara a un aprendizaje adecuado por parte de la red neuronal. De esta manera, todos los valores se encontrarán dentro del rango [0,1].

Gracias al fichero *minMax.*csv, creado y actualizado durante las fases de generación de datos, se conocen tanto el valor máximo como el valor mínimo de cada una de las columnas de los datos.

Un nuevo fichero llamado *data_normalized.csv* será donde se almacenen las filas normalizadas.

Una vez finalizada esta fase, ya se puede entrenar la red neuronal.

5.3.4 Entrenar red

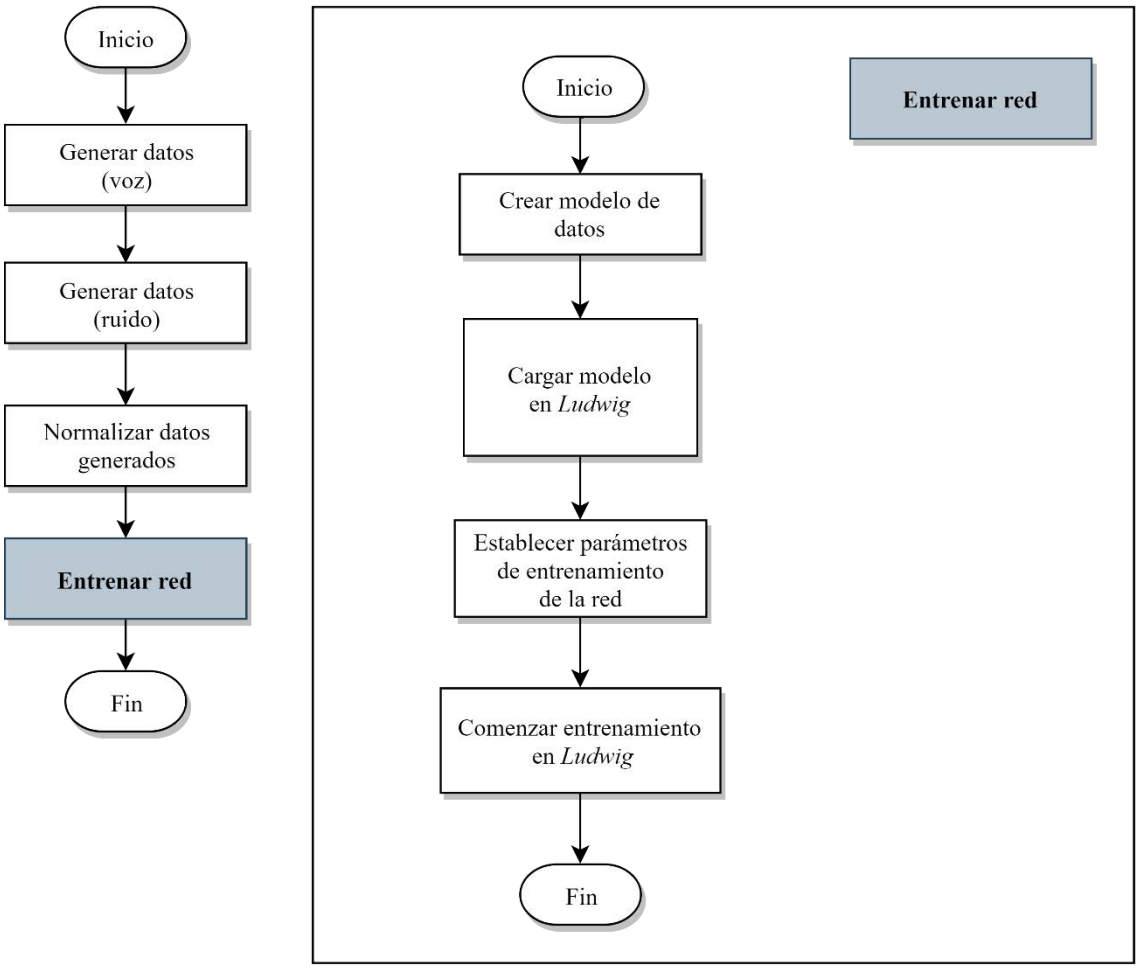

Figura 10: diagrama de secuencia de la fase de entrenamiento de la red

Gracias al uso de la librería *Ludwig*, el proceso de entrenar la red es sencillo. Tan solo se necesita crear un modelo de datos, que consiste en la definición de las entradas de la red (su tipo y nombre) así como de las salidas (en este caso, será solo una salida, *voice*). Todas las entradas, que son las columnas correspondientes a las características espectrales de nuestro fichero *data_normalized.csv*, se modelan como entradas de valores numéricos. La salida *voice* es del tipo binaria y corresponde a la última columna de *data_normalized.csv.*

Ludwig entonces crea una red neuronal utilizando el modelo previamente definido. Los parámetros de entrenamiento de la red serán constantes a lo largo de todas las pruebas, las cuales se especificarán más adelante en el capítulo de Resultados.

Una vez efectuado el entrenamiento, *Ludwig* almacena en el directorio raíz de *data_normalized.csv* la red neuronal para ser utilizada posteriormente en tareas de predicción en el segundo y último modo de funcionamiento del sistema, que veremos a continuación.

5.4 Predicción

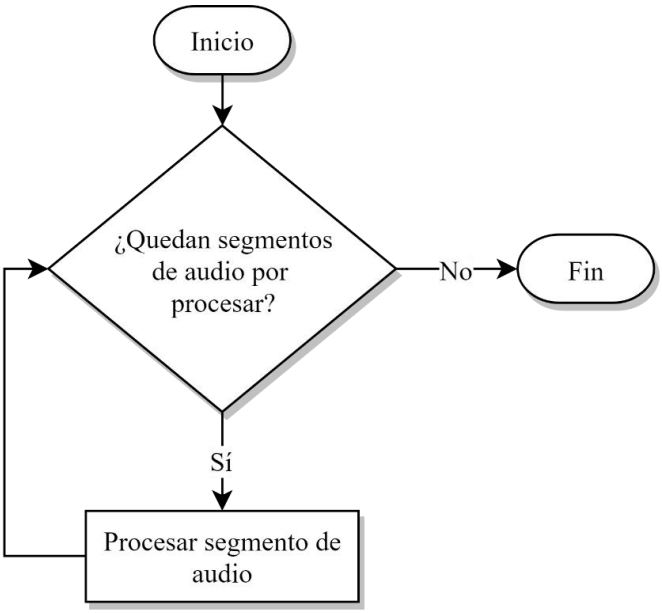

Figura 11: diagrama de secuencia del modo de funcionamiento 2 del sistema

En este segundo modo de funcionamiento se procesa un audio de entrada *frame* a *frame* (es decir, por conjuntos de *samples* del audio). En este sistema se ha implementado de tal manera que la entrada es un fichero de audio seleccionado previamente. Otra opción podría haber sido utilizar como entrada el micro del computador, pero se ha escogido la primera opción para poder utilizar muestras concretas de cara a realizar pruebas consistentes.

En el modo de predicción el procesamiento de audio se realiza mediante una función de *callback*. Esto significa que el sistema, al recibir un *frame*, lo procesa y no lo devuelve al dispositivo de salida de audio hasta acabar. Esto provoca que existan tiempos entre *frame* y *frame* en el que no se está reproduciendo audio. Para solucionar este problema se puede utilizar un *buffer*. De esta manera, al iniciar la ejecución, el sistema acumula los *frames* (procesados) necesarios para que una vez liberado el *buffer* la reproducción sea fluida y no haya secciones vacías en la reproducción. Sin embargo, esta solución no se ha podido

implementar con las herramientas que proporciona *Python*. Este es un motivo por el que la gran mayoría de aplicaciones de audio están escritas en lenguajes como C++ por ser más adecuado para el procesamiento en tiempo real. Entonces, en este proyecto existe el problema de escuchar el audio "cortado" mientras es procesado. De todas formas, se guarda el audio procesado en un fichero *wav* al mismo tiempo que se procesa. De esta manera se puede escuchar posteriormente el audio procesado sin los molestos cortes.

Se ha intentado optimizar el tiempo de procesamiento de cada *frame*, pero no se ha logrado reducir el tiempo lo suficiente para no necesitar un *buffer*. El oído humano es capaz de percibir cortes en el audio de menos de aproximadamente 10 ms, por tanto, si el procesamiento por *frame* tarda más que ese tiempo, el efecto cortado es evidente.

En cuanto al diseño en sí, se ha comprimido todo en una sola fase en este modo de funcionamiento. A continuación, se explicará dicha fase.

5.4.1 Procesar segmento de audio

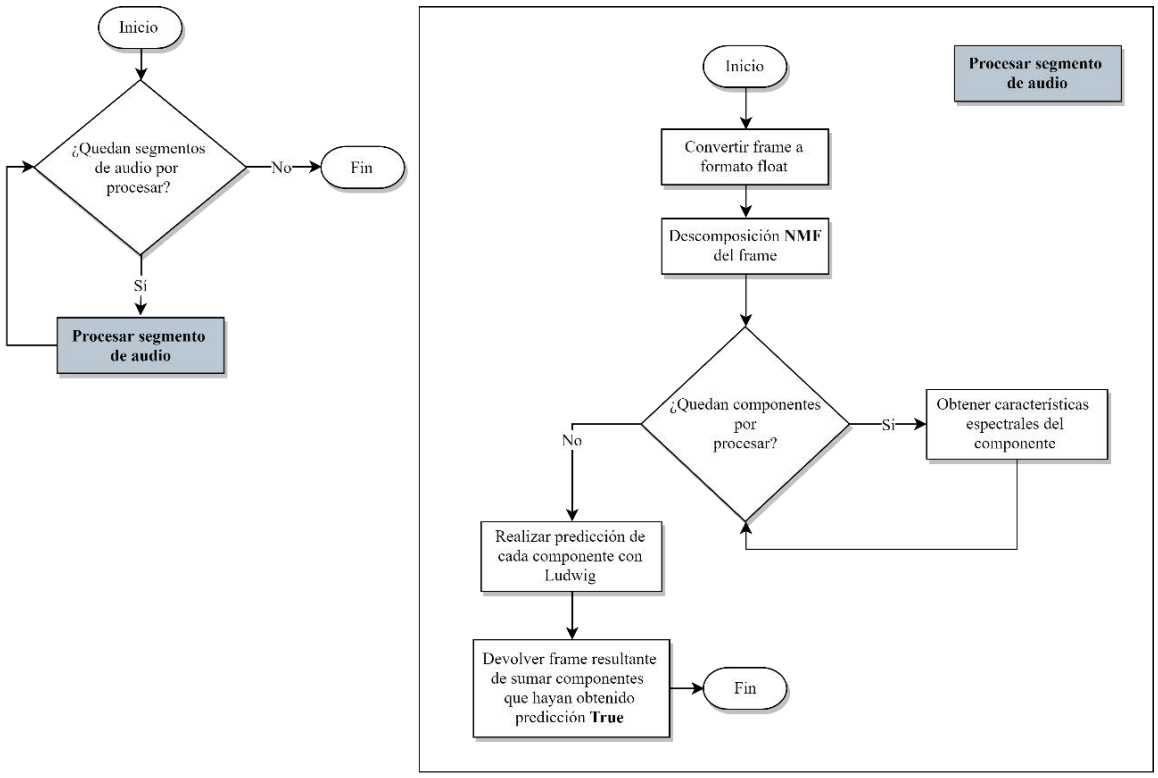

Figura 12: diagrama de secuencia de la fase de procesamiento de un *frame* de audio

El primer paso consiste en convertir el *frame* recibido en formato numérico de coma flotante. Esto es requisito necesario por ciertas librerías como *LibROSA* para poder procesar audio (típicamente, el audio por defecto es de 16 bits). Los siguientes pasos son

similares a los ejecutados en la generación de datos en el modo de funcionamiento 1. La diferencia es que las características se introducen en la red, ya entrenada, en modo de predicción. *Ludwig* recibe los datos por medio de una matriz que contiene los valores de las características para cada componente. Esta vez, *Ludwig* devolverá un vector con una evaluación resultante para cada componente con *True* o *False*.

Sabiendo qué componentes son de voz (valor de predicción *True*), se seleccionan y se suman. Esta señal, resultante de sumar dichos componentes de voz, se devuelve a la función de *callback* para que se reproduzca por el dispositivo de audio y así poder recibir un nuevo *frame* a procesar. El audio, además de reproducirse por el dispositivo de audio del computador, se almacena en el disco duro, progresivamente, *frame* a *frame*, de forma que pueda escucharse posteriormente.

A continuación, se dará por concluido este capítulo abordando el problema de la optimización.

5.5 El problema de la optimización

El código se implementó, desde el principio, para ejecutarse en un único núcleo. Sin embargo, a medida que aumentaba la cantidad de datos a utilizar, se comenzó a necesitar mayor rapidez de procesamiento. Manejar grandes cantidades de datos requería decenas de horas para completar ciertas fases del sistema. Por ejemplo, se necesitaban aproximadamente 6 horas sólo para generar los datos de voz. A ese tiempo ha de sumarse otra cantidad similar para generar los datos de ruido. Además, el proceso de normalizar los datos generados era muy pesado y podía consumir varías horas también.

Puesto que los datos generados varían con los valores de los parámetros escogidos, para cada prueba han de generarse nuevos datos. Esto dificulta enormemente la experimentación debido al coste temporal.

Para aliviar este problema se han seguido una serie de tácticas. Por un lado, paralelizando el código. Por otro lado, optimizando la forma en que los algoritmos estaban escritos. Puesto que se manejan gran cantidad de vectores con la librería *NumPy*, existen diversas formas de escribir operaciones con resultados idénticos, pero con tiempos de ejecuciones decenas de veces más rápidos. Esto es utilizar técnicas de *Python* como *List Comprehension* con el fin de suprimir bucles *for*.

Gracias a la paralelización del código se han logrado aceleraciones considerables en varias funciones. La sección más problemática fue la relativa al modo de funcionamiento de Generación de datos y entrenamiento. Tras la paralelización se ha conseguido un *speed up* de hasta 5 veces en la generación de datos utilizando 6 núcleos en la ejecución del código. Sin embargo, no todos los problemas encontrados son respecto al tiempo de ejecución sino al consumo de memoria.

Por un lado, se ha buscado reducir el consumo de memoria secundaria. Almacenar los resultados de las pruebas, junto a los datos generados para cada una de estas, acaba consumiendo cientos de gigas de almacenamiento. Este consumo se ha reducido de dos maneras. En primer lugar, no se han almacenado los valores que sean cero en los ficheros *csv.* En segundo lugar, se han reducido el número de decimales a almacenar en los datos a 4 dígitos únicamente. De esta forma se ha logrado reducir el tamaño de los ficheros hasta la mitad. Dichas técnicas también ayudan a reducir el tiempo de lectura de las filas o de las columnas del fichero *csv.*

Por otro lado, se han encontrado problemas respecto al uso de memoria principal. Muchas de las funciones de *Pandas,* para el manejo de ficheros *csv,* funcionan cargando el fichero completo en memoria principal. Si el fichero es demasiado grande, acaba produciéndose un error de memoria. Por ello, todas las funciones se han desarrollado de forma que los ficheros *csv* se procesan por pedazos. La librería *Ludwig* tiene la particularidad de convertir el fichero de datos de entrada *csv*, el *dataset*, en formato *hdf5*. Esto se hace porque *hdf5* es un formato más rápido en la lectura de datos de gran tamaño. Sin embargo, *Ludwig* realiza esta conversión utilizando *Pandas*, de la manera anteriormente descrita, intentando cargar el fichero *csv* en memoria principal. Si no se genera el fichero *hdf5* el entrenamiento no se realizará. Para solucionar este problema se ha creado un script (no se entrará en detalle) que crea el fichero *hdf5* manualmente, columna a columna. Es un procedimiento costoso, pero de esta manera algunas pruebas extremadamente pesadas (15 gigas) han podido efectuarse.

En definitiva, la optimización del sistema es un pilar del proyecto, puesto que el éxito o no de este viene en gran medida determinado por la cantidad de información que se pueda manejar y si es bajo tiempos viables.

6. RESULTADOS

6.1 Pruebas realizadas

Se han realizado una serie de pruebas sobre el sistema y se han comparado los resultados obtenidos. En primer lugar, todas las pruebas se han realizado sobre un mismo conjunto de 3 muestras de audio de 10 segundos de duración. Dichas muestras proceden de informativos televisivos.

En la muestra número 1 aparece la voz de la presentadora del informativo además de la música del telediario de fondo. También aparece, sobre la mitad de la muestra, ruido de gente hablando. Los últimos 3 segundos sólo existe la música del informativo y parte del ruido de la noticia que se está mostrando.

En la muestra número 2 es similar a la muestra número 1 con la diferencia de hablar un presentador en vez de una presentadora.

En la muestra número 3 vuelve a hablar una presentadora, pero de fondo hay una canción pop reproduciéndose a un volumen considerable, mucho más alto que la música del informativo que aparecía en las anteriores muestras.

Se ha elegido un conjunto pequeño ya que la evaluación de los resultados, al ser realizada de forma manual, auditivamente, en busca de mejoras, es más fácil realizar la comparación entre las diferentes muestras, dos a dos. Bien es cierto que, en un supuesto caso de poder realizarse las pruebas en un tiempo menor, sería interesante automatizar la evaluación de los resultados. Para ello necesitaríamos disponer de las pistas de voz y de ruido por separado. De esta manera, se realizaría la separación de la mezcla de ambas pistas y se calcularía la semejanza entre el resultado y la pista de voz. Como se especificó en el apartado de Solución Propuesta, podría utilizarse el Coeficiente de correlación de Pearson para buscar la similitud entre ambas muestras. Entonces, las pruebas se realizarían iterativamente en busca de mejorar dicho coeficiente de correlación para todo el conjunto de muestras utilizadas.

Principalmente, las pruebas varían entre sí en el valor de los parámetros principales. Como se mencionó en apartados anteriores, el desarrollo del proyecto no ha sido lineal y se han retornado a fases previas para realizar mejoras y ajustes. Por ejemplo, se ha incrementado el número de *datasets* utilizados con el fin de aumentar la variedad y

cantidad de las muestras. Sin embargo, se ha decidido incluir todas las pruebas ya que han contribuido en la tarea de identificar los mejores valores paramétricos.

No se ha incluido en la tabla, pero se ha observado que a partir de 10 épocas de entrenamiento el aprendizaje comienza a estancarse y ya no compensa continuar con el entrenamiento por el riesgo de producir *overfitting*, esto es, incapacidad de la red clasificadora de generalizar, limitando su capaz de predicción a únicamente los datos con los que se ha realizado el entrenamiento. Este fenómeno se prevé que pueda ocurrir observando que, a partir de cierta época del entrenamiento, la variación en los valores de *accuracy* y de *loss* comienza a ser insignificante (menos de 0.01 puntos).

En la siguiente tabla se muestra la composición de los *Dataset* utilizados para las pruebas:

Dataset Version	**Datasets de Voz**	**Datasets de Ruido**
1	Warden P. Speech Commands, CMU PDA Database	IRMAS, UrbanSound8K Dataset
2	TED Dataset, Warden P. Speech Commands, CMU PDA Database	IRMAS, UrbanSound8K Dataset
3	VoxForge SpeechCorpus	IRMAS, UrbanSound8K Dataset
4	Warden P. Speech Commands, CMU PDA Database, VoxForge SpeechCorpus, TED Dataset	IRMAS, UrbanSound8K Dataset, Drum-machines, NSynth Dataset, Acoustic Event Dataset

TABLA 2: COMPOSICIÓN DE DATASETS PARA PRUEBAS

Como se puede observar, en la última versión se ha incluido el mayor número de muestras. En los *Dataset* 2 y 3 se probó a variar los datos usados de voz para comparar los resultados cuando ocurre esto. Además, a partir del *Dataset* 2, se ha equilibrado el número de muestras de voz frente a las de ruido.

En la siguiente tabla se recogen las pruebas realizadas, donde se indican los valores de los parámetros, así como el porcentaje de precisión de predicción y pérdida de la red. A priori, un mayor valor de precisión (*accuracy*) y menor de la pérdida (*loss*) es deseable. Sin embargo, no es una métrica definitiva y es necesario evaluar el resultado generado para las 3 muestras para determinar si se produce una mejora sustancial respecto a otros valores.

Test Number	Dataset Version	FFT Size (wave samples)	K	Chunk Size (wave samples)	Max. Voice Size (seconds)	Max. Noise Size (seconds)	Accuracy	Loss
1	1	128	16	2048	0.5	0.5	0.828	0.417
2	1	128	16	512	0.5	0.5	0.828	0.422
3	1	128	16	256	0.5	0.5	0.827	0.436
4	1	128	16	128	0.5	0.5	0.826	0.433
5	1	256	16	2048	0.5	0.5	0.843	0.358
6	1	256	16	1024	0.5	0.5	0.837	0.384
7	1	256	16	512	0.5	0.5	0.830	0.414
8	1	256	16	256	0.5	0.5	0.830	0.425
9	1	512	16	2048	0.5	0.5	0.859	0.323
10	1	512	16	1024	0.5	0.5	0.837	0.388
11	1	512	16	1024	0.5	0.5	0.833	0.407
12	**1**	**1024**	**16**	**4096**	**0.5**	**0.5**	**0.874**	**0.285**
13	1	1024	16	2048	0.5	0.5	0.850	0.352
14	1	1024	16	1024	0.5	0.5	0.840	0.391
15	1	2048	16	4096	0.5	0.5	0.865	0.319
16	2	1024	16	4096	0.5	0.5	0.845	0.359
17	2	512	16	4096	0.5	0.5	0.912	0.207
18	2	1024	16	2048	0.5	0.5	0.812	0.431
19	3	512	16	4096	0.5	0.5	0.866	0.304
20	3	512	16	4096	0.5	1	0.891	0.246
21	3	512	32	4096	0.5	0.5	0.831	0.383
22	3	1024	16	4096	0.5	0.5	0.863	0.329
23	3	1024	32	4096	0.5	0.5	0.792	0.451
24	3	1024	64	4096	0.5	0.5	0.715	0.547
25	**3**	**1024**	**32**	**4096**	**1**	**1**	**0.784**	**0.462**

26	4	1024	16	4096	0.5	0.5	0.782	0.461
27	4	1024	32	4096	0.5	0.5	0.737	0.533
28	4	1024	32	4096	1	1	0.723	0.554
29	4	1024	64	4096	1	0.75	0.693	0.574
30	4	1024	96	4096	0.5	0.5	0.700	0.577
31	4	1024	16	4096	2	1	0.796	0.437
32	**4**	**1024**	**16**	**4096**	**4**	**2**	**0.792**	**0.451**
33	4	1024	96	4096	2	1	0.669	0.597

TABLA 3: PRUEBAS REALIZADAS

El tamaño de ventana de la Transformada de Fourier (*FFT Size*) indica el número de *samples* de audio que se utilizan para calcular dicha transformada. A mayor valor de este parámetro, mayor número de frecuencias analizadas.

El siguiente parámetro, *k*, es el número de componentes NMF en los que se descompone la señal a la hora de realizar la separación. Se ha experimentado con valores más altos a medida que se ha avanzado en el proyecto. Por lo general, el incremento de este valor provoca una caída en la precisión (*accuracy*), aumento de la pérdida (*loss*) y un incremento del coste computacional. No se puede afirmar con certeza que el incremento de este valor mejore la separación. Es cierto que suele generar mayor suavidad en el resultado, pero suele ser a costa de introducir ruido.

El tamaño de porción (*chunk size*) establece el número de samples en que se divide cada porción de la señal. Se ha encontrado que valores de *chunk size* pequeños provocan que la señal procesada suene "cortada". El *chunk size* es el tamaño o número de *samples* en los que se divide cada muestra a procesar. Este tamaño, consecuentemente, es el mismo que utiliza el sistema para procesar audio en tiempo real, es decir, procesa el audio de entrada en porciones del mismo tamaño que se ha usado para la fase de generación de datos y entrenamiento. Analizando las pruebas, los mejores resultados obtenidos tienen un valor de *chunk size* mayor que 512. De esta forma desaparecen los "cortes" en el audio procesado y se mejora el resultado de la predicción.

Los dos últimos parámetros establecen la cantidad de información que se utiliza de uno u otro *Dataset*. Establece la duración máxima a cargar para cada muestra de su respectivo

conjunto. Gracias a este parámetro, se puede evaluar de qué manera afecta al resultado proporcionar los datos de una u otra forma. Por lo general, se ha buscado el equilibrio, 50-50, aunque se han probado otras combinaciones.

6.2 Evaluación de los resultados

Para facilitar la comparación de las muestras, se adjuntará un enlace a través del cual escuchar dichas comparaciones.

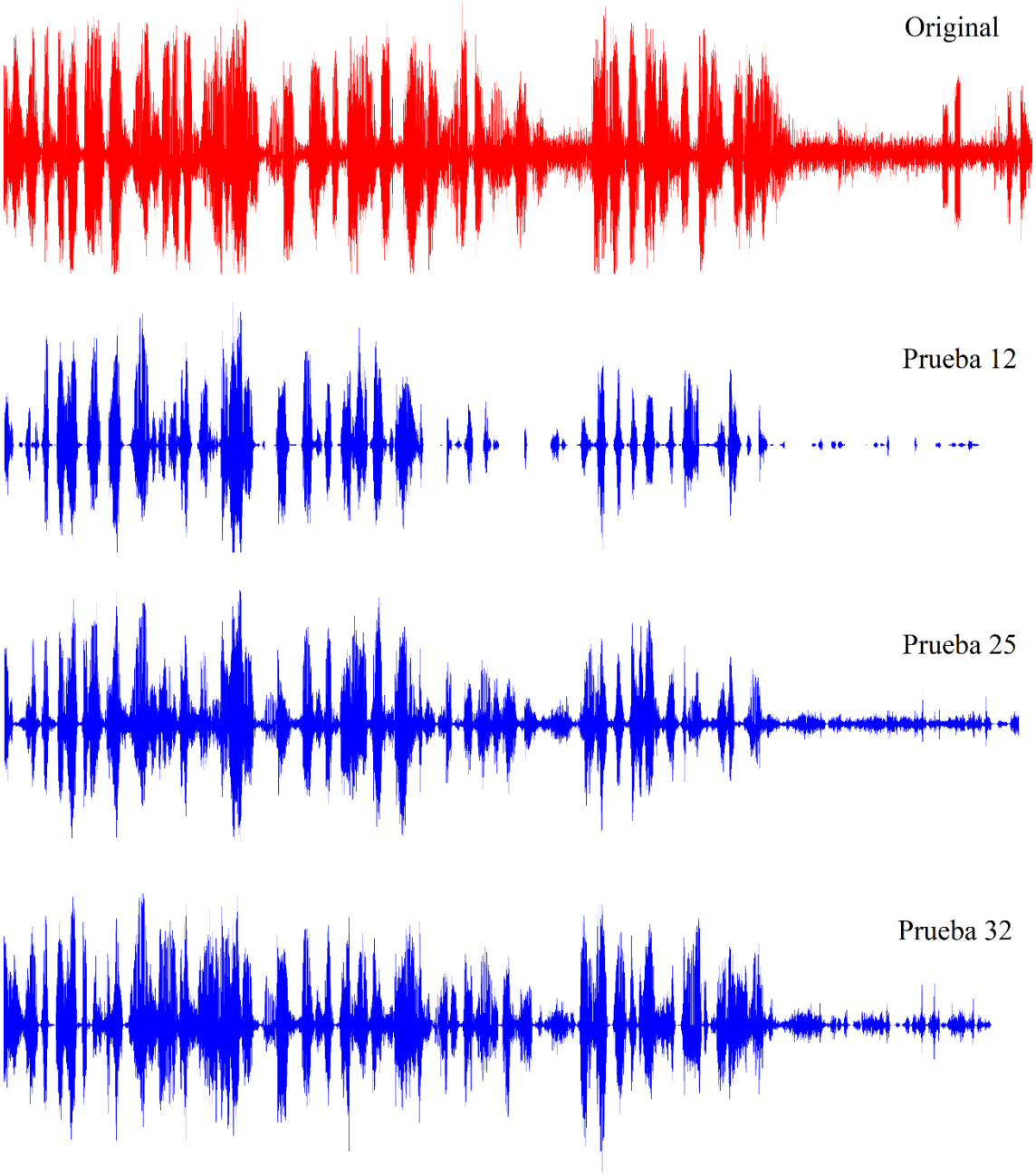

Figura 13: Comparación de resultados en la muestra 1

En la figura 11 se muestra la comparación de los resultados de procesar la muestra 1 en las pruebas 12, 25 y 32. Puede escucharse dicha comparación en este enlace[3].

Para la prueba número 12 se ha conseguido un 87.5 % de precisión en el entrenamiento de la red. Es un número mayor que el alcanzado por las otras dos pruebas. Sin embargo, esto no es garantía de que la separación sea mejor.

Los datos utilizados, así como los valores de los parámetros son muy relevantes en el resultado obtenido. Las tres pruebas tienen en común un valor de ventana FFT de 1024 *samples* y un valor de *chunk size* de 4096 *samples*. Se ha llegado a la conclusión de que estos valores son los más efectivos a medida que se han ejecutado las pruebas.

A simple vista se observa que en la prueba 12 el sistema es capaz de aislar las secciones principales de la voz. Sin embargo, hay algunas palabras que han perdido información esencial hasta el punto de sonar débiles y difícilmente reconocibles. Por otro lado, se observa mayor dificultad a la hora de recuperar sonidos de consonantes frente a sonidos de vocales. Esto puede deberse a la similitud entre sonidos de consonantes con otra serie de ruidos. Una hipótesis que justificaría el primer problema y posiblemente el segundo, es la necesidad de entrenar la red con mayor cantidad de información. Sin embargo, en las siguientes muestras se ha utilizado mayor cantidad de información, especialmente en la prueba 32, que utiliza la versión 4 del *Dataset*, y se comprueba que sí, en efecto, se recupera mayor cantidad de información de la voz, pero a coste de introducir información de ruido.

Analicemos ahora los resultados para la segunda muestra, los cuales pueden escucharse en el siguiente enlace[4].

[3] https://soundcloud.com/francisco-javier-alonso-rubio/sets/muestra-1-comparacion/s-2lVcA
[4] https://soundcloud.com/francisco-javier-alonso-rubio/sets/muestra-2-comparacion/s-KVtEL

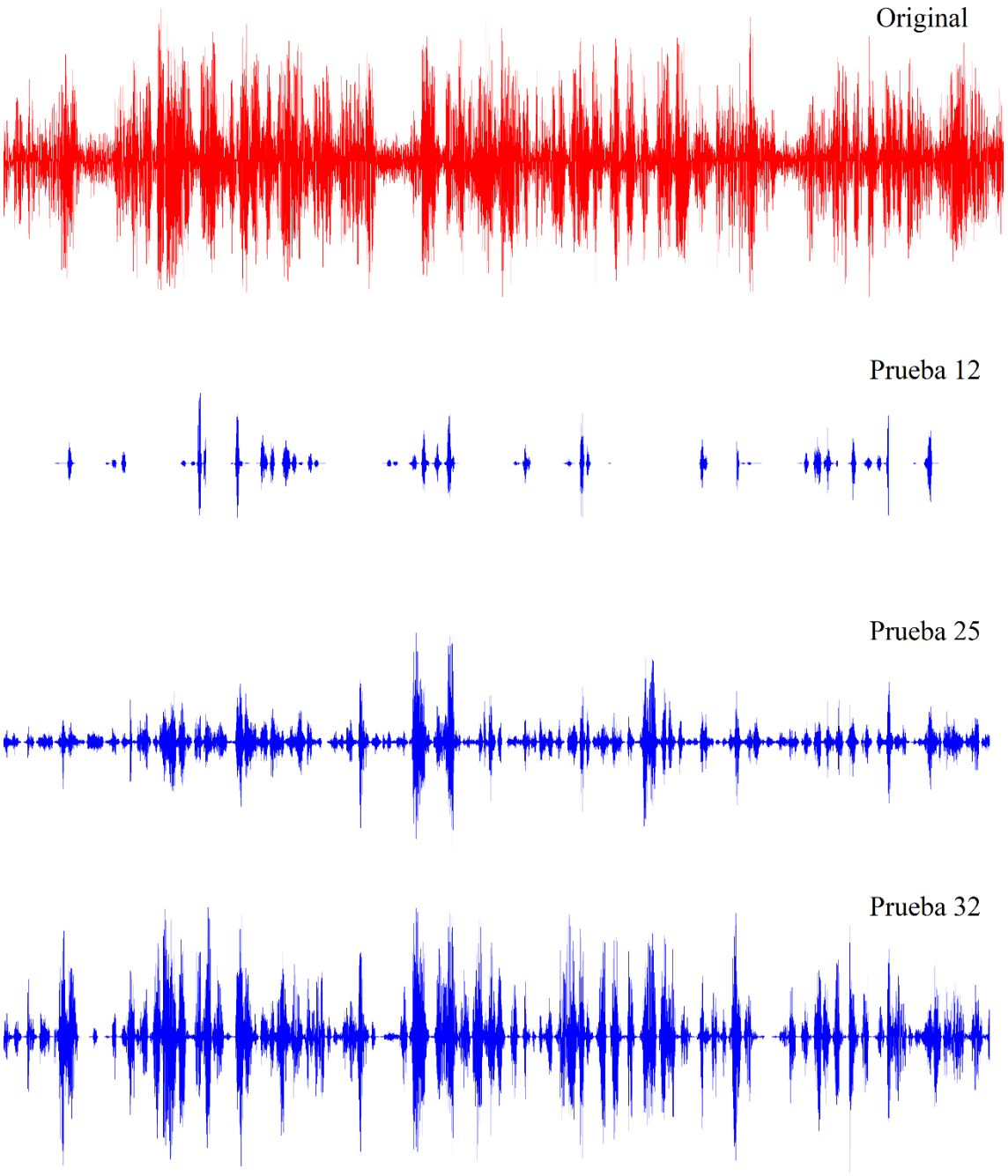

Figura 14: Comparación de resultados en la muestra 2

La separación ha sido muy deficiente en la prueba 12. En este caso, a diferencia de la anterior muestra, o como veremos a continuación, en la última muestra, se trata de un interlocutor masculino. Este puede ser el motivo de la separación tan deficiente. Puede deberse a un sesgo en los datos de voz que contengan más muestras de voz femenina. También puede ser por parte de las muestras de ruido, que presenten similitudes con esta voz masculina en concreto. En cualquier caso, la separación es más efectiva en posteriores pruebas.

Los resultados de la última muestra pueden escucharse en este enlace[5]. Esta muestra contiene música a un volumen mayor, más predominante en la mezcla que en muestras anteriores.

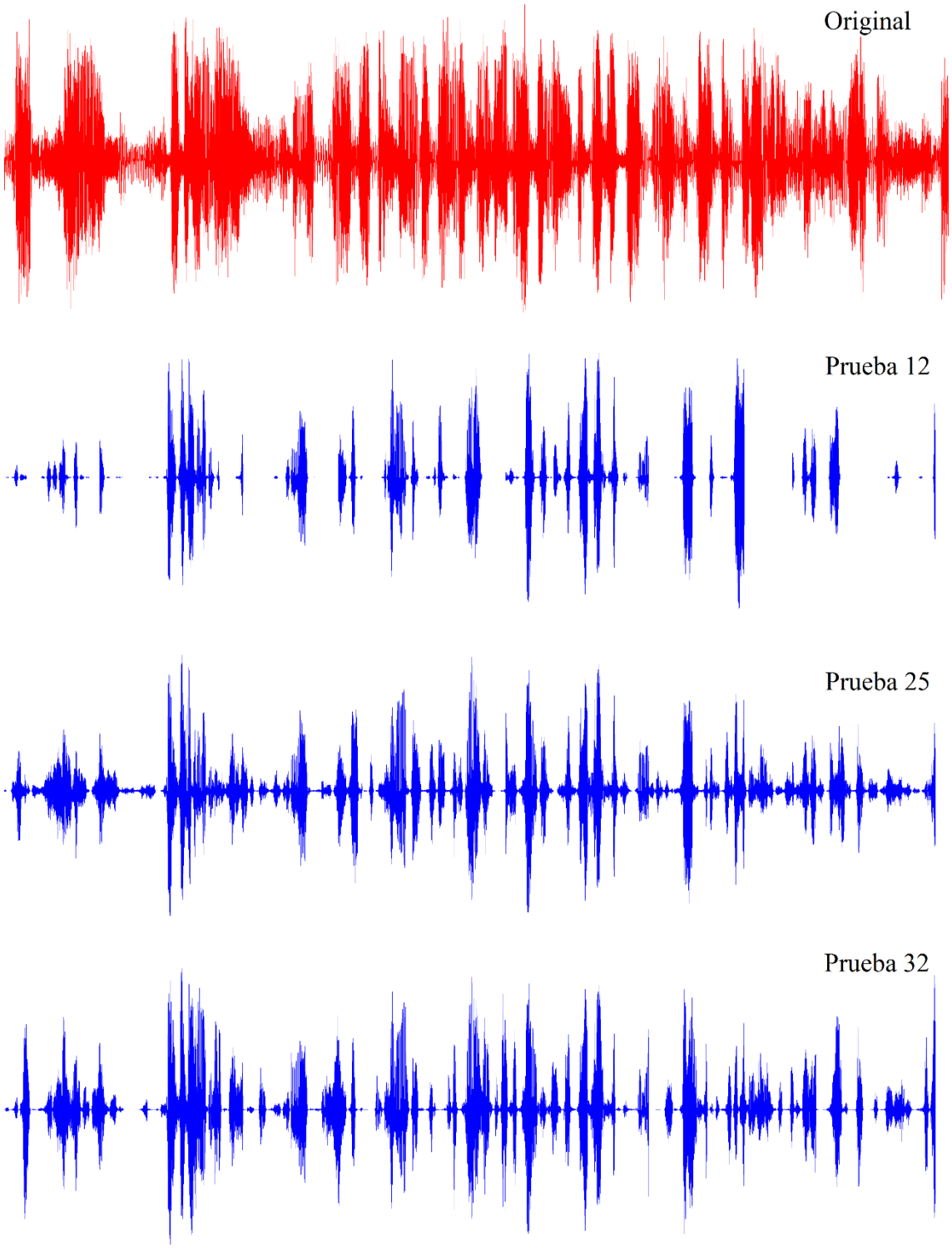

Figura 15: Comparación de resultados en la muestra 3

[5] https://soundcloud.com/francisco-javier-alonso-rubio/sets/muestra-3-comparacion/s-vlFVc

En la prueba 12 vuelve a encontrarse el problema identificado en la muestra 1, la falta de información recuperada. Las posteriores pruebas lo que consiguen es, principalmente, llenar esos vacíos de información que el sistema genera en la prueba 12. Si se escucha la prueba 32, puede comprobarse que tanto la voz del interlocutor como del cantante se conservan. Esto es importante ya que el sistema no busca separar al locutor predominante, sino que recupera todo aquello que reconoce como voz. El resultado en esta última prueba es aceptable, pero hay frecuencias (medianas y altas) de la caja y del bombo de la batería que siguen sonando. Puede apreciarse que el bajo y las frecuencias bajas del bombo han desaparecido prácticamente en su totalidad.

En el siguiente capítulo se analizarán las conclusiones extraídas de estos resultados.

7. CONCLUSIONES

Para finalizar el documento debemos recordar los objetivos establecidos en el primer capítulo. El objetivo principal se ha cumplido, parcialmente. La separación de voz no se logra con efectividad y existe margen de mejora en el sistema. Considero que el sistema se encuentra en un estado más cercano a lo experimental que a ser usado en el mundo real o de forma comercial.

En cuanto a los objetivos secundarios, el funcionamiento en tiempo real se cumple hasta cierto punto. Como se ha explicado en el capítulo de Implementación del sistema, se requiere de introducir un *buffer* donde almacenar los *frames* de entrada, ya procesados, para poder generar un *stream* de audio fluido. Una solución podría comenzar por implementar este sistema en un lenguaje como C++, ideal para programas de audio.

El segundo objetivo secundario considero que se cumple. El sistema funciona, con mayor o menor efectividad, igual en cada instante, sin la necesidad de especificar previamente información sobre la naturaleza del audio a procesar.

El último objetivo secundario se cumple gracias a implementar el sistema en el lenguaje *Python*. Pese a las carencias que presenta a nivel de rendimiento, en concreto en la sección de procesamiento en tiempo real, nos permite ejecutar el código en diversos sistemas operativos. Prueba de ello es que partes del desarrollo se han hecho en Linux y otras en Windows.

A continuación, se plantean una serie de posibles mejoras:

En primer lugar, se ha comprobado que los datos utilizados son muy importantes en el resultado obtenido. Debe asegurarse el uso de un *Dataset* equilibrado y de calidad, que contemple un gran rango de posibilidades, especialmente en cuanto a las muestras de voz.

En segundo lugar, se puede experimentar con diferentes topologías de la red neuronal en busca de mejores valores de *accuracy.* Por ejemplo, utilizar redes neuronales recursivas.

En tercer lugar, se ha planteado la idea de utilizar *n* redes neuronales clasificadoras. Es decir, se plantean *n* categorías de ruido, donde cada categoría corresponderá a una red neuronal diferente y a la hora de realizar la separación, se evalúan las predicciones obtenidas entre las *n* redes neuronales. Esta idea se encuentra implementada en el código, de tal forma que si se desea se puede usar esta funcionalidad o no. Sin embargo, al observar que no se producen mejoras con respecto al anterior sistema, incluso se observa un empeoramiento de los resultados, se ha decidido tan solo mencionar esta idea como una posibilidad a mejorar. La idea principal es que el uso de diferentes redes neuronales facilitaría el entrenamiento individual de estas, ya que los datos de entrenamiento serían más específicos y no tan generales. Otra idea, en cierto modo inversa, sería realizar la categorización esta vez en la voz, en hombres y mujeres, o incluso por edad. Otra posibilidad sería establecer una estructura jerárquica de redes neuronales clasificadoras, donde a medida que se desciende en la jerarquía se busque clasificar los componentes en categorías cada vez más específicas. En cualquier caso, indagar en estas posibilidades implica un esfuerzo que sobrepasaría, a criterio personal, el esfuerzo esperado de un TFG.

En conclusión, esta área de investigación está completamente vigente y abierta a posibilidades de mejora. La ausencia de un consenso en cuanto al Estado del Arte dificulta la obtención de progresos, globales, en esta área. Además, tampoco ayuda que gran parte de las tecnologías desarrolladas se encuentren no sean *open source*.

Afortunadamente, gracias a la mejora y aumento de los *datasets* disponibles, así como de los avances en redes neuronales, es posible que dentro de pocos años lleguemos a disponer de una solución consensuada y, a partir de ese momento, centrarnos en mejorar dicha solución.

8. ENGLISH SUMMARY

8.1 Introduction

The voice is one of the most primitive communication mediums we have available. Since our early beginnings, from a practical point of view, we used it to communicate with beings like other humans or even animals.

But in the middle of the XXI century, thanks to the advances in computing, a new possibility arises: communication with machines using the human voice. Machines became receptors, human voice interpreters.

In the last fifteen years, great developments were made in this field. A great amount of applications of different purpose were developed that they use human voice as a communicative input:

Personal assistants (Alexa, Siri), conversational bots (Mitsuku), text-voice transcribers, (Google Transcription), translators (Translate Voice), biometric systems (VoicePIN), surveillance systems (ECHELON), videogames (Oculus Voice), among others.

All these applications have in common the need that the voice comes into the system in the best shape possible. This way, applications will perform optimally.

The human voice isn't the only habitant of the sound spectra, but it coexists, simultaneously, with a great variety of different sounds. This produces the phenomena known as "voice masking", this means, difficult on recognizing the human voice. Under this context, solutions must be found to mitigate this problem.

8.2 Objectives

The main objective is to develop a system that, setting an audio as input, processes it in a way that only the human voice remains, discarding any other source or sound not identified as human voice.

Other objectives are considered, although secondary, in this project:

- The system should work in real time, so the processed audio is retrieved after just a short delay.

- The system should work in the same way in the first instant as the last of the input audio. This way, the system will have the same effectivity in every instant.

- The system should be, preferably, multiplatform, so it can be run under operative systems as Windows or based on Linux.

8.3 Regulatory Framework and Social-Economical Environment

Although the system created in this project won't be commercialized based on the results obtained, some considerations should be taken. Firstly, in case of a possible, future commercialization, all the software (libraries) and datasets licenses need to be analyzed.

Luckily, all software used in this project is open source and there are no restrictions whatsoever in case of commercialization. Most of the licenses are of Apache 2.0, MIT or BSD types. Still, datasets need to be considered, but it was hard to determine the conditions they are under in case of commercialization, because the data was usually presented along some developed software (which wasn't used or needed in any case), which had itself a license of the previous types.

Also, if we think in terms of the Social-Economical impact this kind of system would have we need to consider the fact that this software would probably exist only as a complement, addition or plugin to other bigger software. This system is more like a tool than a software itself. This means that in terms of commercialization a library or plugin format would be preferable. Returning to the point of measuring the Social-Economical impact of this now called "tool", we would need to consider in which kind of applications it would be integrated. Some applications were mentioned in the introduction, but all of them have way different targets and use cases. For example, some could be categorized as governmental applications and others as private use applications, even as videogames. This makes very difficult to foresee possible impacts in society and economy, as all these applications work under completely different environments.

8.4 State-of-the-Art: Existing solutions

The problem we are facing is a problem of source separation. A source, in the context of sound, could be defined as the origin of a set of audio signals. A source could be a person, a guitar, a car, a violin, a dog, etc. Each of these sources generate different sets of distinguishable audio signals (although it could happen that different sources produce

similar sounds). In this project we focus on human voice sources, considering any other sources as if they were noise.

In the audio processing field, a popular set of techniques related to this problem are part of the so-called Blind Source Separation concept. Roughly speaking, it consists on performing audio source separation with limited, or none, previous information about the nature of the sources present in the sound where the separation is performed. To perform the separation a technique such as PCA [1], ICA [1] or NMF [2] is used.

Nonetheless, it's difficult to identify the State-of-the-Art. There isn't a consensus, or a solution recognized as the best. This problem is partially caused by the non-open source nature of most of the applications in this field. Some of the main applications are *Audionamix XTRAX STEMS 2*, *iZotope RX 7* or *AudioSourceRE*. Meanwhile, the first open source application focused on audio source separation, *openBlissart*, was released in 2011. Other audio source separation open source packages were released in later years such as *untwist* (2016) or *Nussl* (2018). But, still, there isn't a State-of-the-Art open source solution [9].

Another relevant factor in this problem is the fact that most of the proposed solutions are based on *Deep Learning* and it causes the need of big amounts of quality data to perform well. Create a *dataset* of sound samples is a difficult task which is also difficult to automatize. So, years were needed to pass until we reached a point where now there are open access sound *datasets* which satisfies past needs.

The main technique to be used in this project is NMF, Non-Negative Matrix Factorization. NMF was decided to be used, mainly, by three motives:

1) Its popularity has increased in recent years.

2) It has proven to be effective in different situations [2] [4] [5].

3) More intuitive than other existing techniques.

8.5 State-of-the-Art: Non-Negative Matrix Factorization (NMF)

Given a matrix A, factorize this matrix consists on express this matrix as the product of two matrix:

$$A \approx BC$$

Known matrix factorization techniques are SVD, QR o NMF. The last one, NMF, has the particularity that it only uses nonnegative data matrixes. In NMF the usual notation is as follows:

$$V \approx WH$$

Where *V* is the matrix of original nonnegative data. Each column is a sample of the data and each row corresponds to a data characteristic. The spectrogram of the audio to decompose will be used as the *V* data. Using the spectrogram's magnitude is ideal as it is made of nonnegative values.

W is known as the dictionary elements, which is a matrix made of *k* basis vectors. Each column is a basis vector. Lastly, *H* is the weights or gains matrix. Each row represents the gain of the corresponding basis vector.

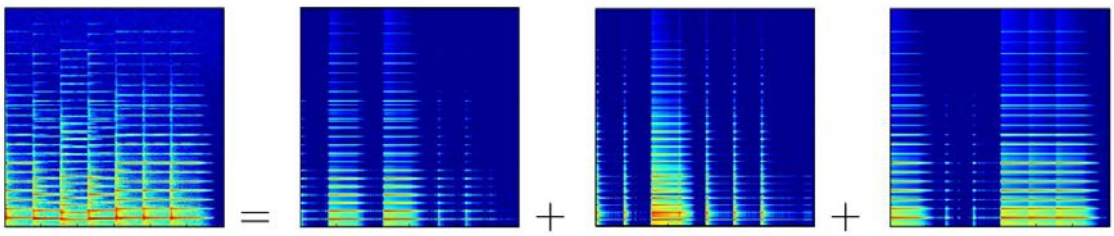

Figure 3: V approximation as the sum of *3* components [3]

The components obtained after the decomposition, summed, will generate an approximation of the original *V matrix.* The number of components needed to recompose *V* is essential because a low number will cause artifacts as information losses. Typically, 16 components are enough to perform reconstruction in all kind of complex signals.

8.6 Proposed solution

At this point we know that NMF is a useful technique to perform the decomposition of an audio signal into *k* components. So, this way, selecting and adding certain components we could retrieve a single source and the same process could be done to retrieve the rest of components present in an audio signal.

The final decision of NMF also comes motivated by the experimental practice, not only by what was found in the literature. A test was performed using two separated audio signals, being a kick drum loop one of them, and a piano loop being the other. The mixture was obtained summing both signals and later NMF was performed. This way, the mixture

was decomposed into *k* components, in this case 16, so each one of them was evaluated by converting them into audio back from their spectrogram info (remember that the data of each component is a spectrogram and, consequently, needs to converted to audio). This evaluation was performed by listening, where some components were found to be part of the piano and others part of the kick drum. From this experiment it was found tow things. In first place, that NMF works (at least in relatively simple separation tasks) and that its easier to perform separation by just selecting the components that are probable to come from one certain source, not taking the nature of the other source into account. This last find is a key heuristic in the design and development of the project.

Knowing that NMF could be a solution capable of separating an audio signal into components to then group them into their different, corresponding sources, the remaining task would be to do this grouping automatically, without human intervention. A neural network will be chosen solution to try to perform this grouping or classification. The network would perform classification using the heuristic stablished before, by voice or noise, being the last category, any other sound not detected as human voice.

The next step is defining the type of data the classifier neural network is going to use. This data will consist of a series of spectral characteristic found in the components extracted from the audios of different datasets, specifically, voice datasets and noise (anything that is not human voice) datasets. Spectral characteristics are, as the name indicates, characteristics obtained from the frequencies present in the audio spectra. This is instant information and not temporal information. To obtain the audio spectra, FFT or Fourier Fast Transform must be performed over the audio signal. The resulting spectra allow us to analyze the audio signal in terms of frequencies and magnitude of each of these frequencies.

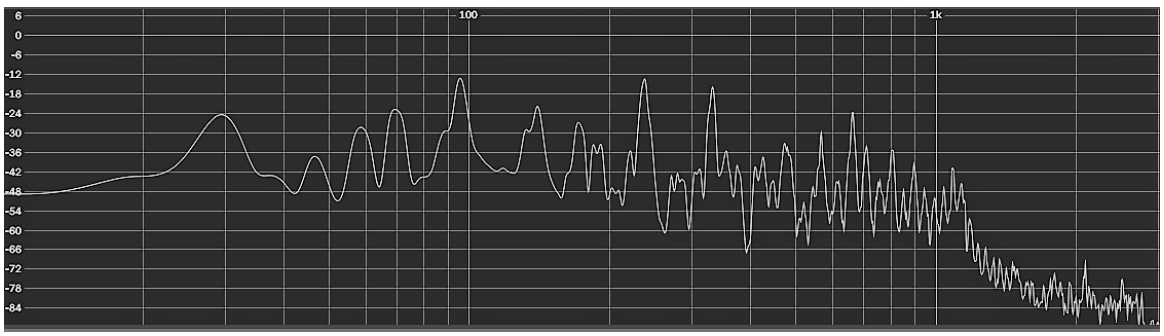

Figure 4: audio spectra in a specific instant

The spectral characteristics used in this project will be the following:

1) **Mean Magnitude of Frequency Sets (MMCF)**: this is a characteristic that calculates the mean magnitude for different bands or groups of frequencies of the audio spectra. In other words, the frequencies are grouped into same sized groups and then performed the mean of their magnitudes. This way we reduce or compress the spectra information into less bands than using all of them, separately.

2) **Mel Frequency Cepstral Coefficients (MFCC)**: this is a widely spectral used characteristic in audio voice applications. It is considered state-of-the-art in terms of audio characteristics.

3) **Spectral Flatness**: these characteristic measures how flat the audio spectra is. Values of the characteristic closers to 1 means that the spectra is closer to being flat, this is, being noisy, and values closer to 0 means that the spectra presents some kind of harmonic structure.

4) **Spectral Centroid**: the last spectral characteristic used is the centroid. This gives the centroid of the spectra, meaning, the frequency where most energy tends to converge.

In resume, the proposed solution will consist on the following:

NMF decomposition will be performed to different voice and noise datasets. The components obtained will be processed so the previous spectral characteristics are obtained from each one of them. Then, these values would be stored in a *csv* file, resulting in our own dataset, which we will use to train a neural network as a NMF component classifier of 2 categories, noise and voice.

Consequently, this system will have two different operating modes, although related as the first mode must be executed before to make Prediction mode available:

1) Data Generation and Training

2) Prediction

8.7 Datasets preparation

Before implementing the system, datasets need to be collected and prepared to use by the system. Typically, voice datasets are composed of words or phrases read by different people. There are some variations in terms of age and sex in the subjects present in the datasets.

For the noise datasets, two key categories were made: Urban Noise datasets (alarms, hits, vehicles, animals, noises in the typical use of the word) and Instruments (drums hits, synthesizers, guitars, pianos, violins, etc.)

All these datasets were applied some preprocessing with the program *WavePad* before being used by the system. Typically, information below 250 Hz was cut in the voice samples. Then, all the samples, whatever the category is, are normalized to 0 dB and then if clicks are founds in the sample they are removed.

8.8 Implementation

Having all the datasets prepared, the solution proposed before is ready to get implemented. The implementation, as mentioned before, will be segmented into two operating modes, Data Generation and Training, and the other mode, Prediction. Nonetheless, these operating modes will share a great number of functionalities between them. Also, the parameters values will be the same between operating modes for each test performed to work as intended.

As mentioned before, the main and only programming language is *Python*. The key external libraries used are *LibROSA*, *SciPy*, *Ludwig*, *Tensorflow*, *PyAudio*, *Pandas* or *NumPy*. *LibROSA* provide implementations for extracting all spectral characteristics. *Ludwig*, which is based on *Tensorflow*, is used to implement the neural network. Lastly, *PyAudio* is used to implement the real time processing part.

Also, a big part of the efforts of this project were used into optimizing the system in two ways. Firstly, in reducing the data generation processing times, mainly by parallelization and code rewriting, and by reducing main and secondary memory usage. Great improvements were achieved this way, allowing to perform more data-heavy tests. It was concluded that this kind of projects, based on operating with large amounts of data, need to focus on optimization tasks to succeed.

Some problems were found in the Prediction operating mode, mainly because of "chopping" audio being played by the sound device after processing the input audio. A buffer is determined to be needed to solve this problem, but *Python* and *PyAudio* don't provide the tools needed to do this.

8.9 Results

Around 30 tests were performed over a set of the same 3 audio samples looking for the best separation results. These 3 samples are 10 seconds long each of them and where obtained from some TV's news program. In two of them the presenter is a woman and in one is a man. In all the samples music or noise will be found, typically sounding at the same time, as the presenter's voices. A small sample size was chosen to facilitate the comparison between the results obtained as it is a manual process, made by ear and by waveform visual analysis.

The tests vary between them in the values chosen for the established parameters. As the development wasn't lineal, some tests were performed using different datasets as input, but the tests remain in the document because the optimal parameters' values were found since the first test until the last one performed, even if the dataset was different, being all the tests useful to the project. The dataset used for the tests differ in content resulting in 4 different versions used of this dataset, made of different datasets itself.

The network training parameters were all the same between tests. The number of training epochs was 10 as more weren't needed. More than 10 epochs would start causing overfitting in the network (incapability of generalization) as the learning would become insignificant (as observed by the small difference in accuracy / loss between epochs). The accuracy and loss obtained by the network could be the main results when comparing different tests, but the resulting processed audio is always the main decider when evaluating the system's performance with the given parameters.

The system parameters used at the tests were the following:

1) **FFT window size**: this is the number of samples used to perform the FFT. When this value is increased, the number of resulting frequencies described is increased. This increases the number of MMCF columns in the generated data.

2) **NMF's *k* value**: this is the number of components that are decomposed the signal into when performing NMF, as seen in the State-of-the-Art section. Bigger values

were used when the project was in a more advanced state. Generally, the increasement of this value causes reduction in the network's accuracy and increases its loss. Still, the increase of this parameter's value caused some kind of "smoothing" in the separation, but at cost of introducing some noise. So, the increasement of this value can't be taken as an improvement and the processed audio needs to be evaluated.

3) **Chunk size**: this parameter sets the amount waveform samples in which the audio sample is split with. It was found that small chunk sizes cause that the processed audio sounds "chopped". This size is the same used as the frame size in the real-time mode of the system. This mean that this parameter uses the same value in both modes, in the mode of data generation and training and the prediction or real time mode.

4) **Max voice size**: maximum seconds duration to load for each voice sample found in the data generation functioning mode.

5) **Max noise size**: maximum seconds duration to load for each noise sample found in the data generation functioning mode.

All the tests were analyzed and compared between them, but in this document, we focus on three specific tests, numbers 12, 25 and 32. We find that the best FFT window size is 1024 and the best chunk size is 4096 across all tests, giving the best results.

Comparing the results across these 3 tests, when the dataset size used to generate the data to train the network is increased, the accuracy decrements and the loss increases. Still, the results get slightly better than using fewer data. In the test number 12 a nice indicator that shows potential in this method is that some key voice cluster are found and isolated. Still, some voice info and details (specially on consonant sounds) are lost. Further tests such as number 25 and number 32 retrieve more voice information, although some noise is reintroduced.

8.10 Conclusions

To finish the document some recapitulation is made, starting with the objectives stated on the Introduction section. Firstly, the main objective is "to develop a system that, setting an audio as input, processes it in a way that only the human voice remains, discarding any other source or sound not identified as human voice". This objective is partially achieved

because the system is capable of “detect” or retrieve some key clusters of voice. Still, details and some other words are lost in the process. This leaves the system in a still experimental state, not ready for a possible commercialization.

The other secondary objectives were the following:

- The system should work in real time, so the processed audio is retrieved after just a short delay. This would work perfectly if Python audio libraries would allow to implement a buffer to hold a few processed frames before releasing them into the output audio device. This would fix the “choppy audio” effect when processing audio in real time. C++ would be a nice alternative with which reprogram the system to introduce this needed buffer and increase the overall performance.

- The system should work in the same way in the first instant as the last of the input audio. This is achieved, because the system is implemented in a way that the use doesn’t need to insert previous info about the audio to process. In resume, the system doesn’t need any kind of previous info about the audio that is processing and works the same all the time.

- The system should be, preferably, multiplatform, so it can be run under operative systems as Windows or based on Linux. This is achieved by implementing the system in the Python language, which offers great compatibility between computer systems (this compatibility depends on the external libraries).

Also, some improvements of the current system are proposed. Firstly, the dataset used could find some improvement and refinement. This is important because the data used is a great factor that separates a successful project from an unsuccessful one.

Secondly, the neural networks could get bigger improvements, starting by using different topologies. Recursive neural networks could be a nice possibility to explore in search of better neural network classifiers.

Thirdly, using more than one neural network could be an interesting idea. These neural networks could be used in different ways and placements. For example, we could set n noise categories so each one has its own neural network classifier trained (with that noise

category data and the voice data, which is common for all). This way the training is more specific so the chances that the accuracy increases for the separated networks compared to a single, generalized one, are greater. Another possibility would be to use these networks in a hierarchy system, where evaluations are refined and refined, trying to fit the components into noise categories, deeper and deeper. The last idea would be to use an opposite point of view, where the categories are stablished on the voice and not in the noise. This way, a category of voice data could be man, another category woman, by age, etc. Still, the effort that would be needed to spend into exploring these ideas would overpass the expected of a Final Year Project.

In conclusion, this field is completely active now and open into greater developments. The absence of a state-of-the-art consensus damages the easiness the achievement of global developments. The fact that still big part of the products released, at least partially, with this purpose, are fewer than the open source ones, damages the overall progress too.

Fortunately, thanks to the increase and development of available datasets and recent developments in neural network technologies, it's possible that in a few years we will find a globally approved state-of-the-art solution, development will part from this solution and will be quicker and, in result, more effective.

9. REFERENCIA DE DATASETS

[D1] "Warden P. Speech Commands: A public dataset for single-word speech recognition, 2017. Available fromhttp://download.tensorflow.org/data/speech_commands_v0.01.tar.gz".

[D2] "CMU PDA Database, recorded by Yasunari Obuchi, 2003. Available from http://www.speech.cs.cmu.edu/databases/pda/index.html".

[D3] "VoxForge SpeechCorpus Audio, 2006, Available from http://www.repository.voxforge1.org/downloads/SpeechCorpus/Trunk/Audio/".

[D4] Bosch, J. J., Janer, J., Fuhrmann, F., & Herrera, P. "A Comparison of Sound Segregation Techniques for Predominant Instrument Recognition in Musical Audio Signals", in Proc. ISMIR (pp. 559-564), 2012.

[D5] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders." 2017.

[D6] "drum-machines, 2008, Available from http://www.hexawe.net/mess/200.Drum.Machines/".

[D7] J. Salamon, C. Jacoby and J. P. Bello, "**A Dataset and Taxonomy for Urban Sound Research**", 22nd ACM International Conference on Multimedia, Orlando USA, Nov. 2014.

[D8] "Acoustic Event Dataset, Available from https://data.vision.ee.ethz.ch/cvl/ae_dataset/ ".

10. BIBLIOGRAFÍA

[1] HYVARINEN, Aapo; KARHUNEN, J.; OJA, E. Independent component analysis and blind source separation. 2001.

[2] VIRTANEN, Tuomas. Monaural sound source separation by nonnegative matrix factorization with temporal continuity and sparseness criteria. *IEEE transactions on audio, speech, and language processing*, 2007, vol. 15, no 3, p. 1066-1074.

[3] Nicholas Bryan, Dennis Sun, and Eunjoon Cho, "Single-Channel Source Separation Tutorial Mini-Series" [online] 2015.

[4] LUDEÑA-CHOEZ, Jimmy; QUISPE-SONCCO, Raisa; GALLARDO-ANTOLÍN, Ascensión. Bird sound spectrogram decomposition through Non-Negative Matrix Factorization for the acoustic classification of bird species. *PloS one*, 2017, vol. 12, no 6, p. e0179403.

[5] SCHULLER, Björn, et al. Blind enhancement of the rhythmic and harmonic sections by NMF: Does it help?, En *Proc. Intern. Conf. on Acoustics (NAG/Tagungsband Fortschritte der Akustik-DAGA 2009), Rotterdam, The Netherlands*. 2009. p. 361-364.

[6] RICKARD, Scott. The DUET blind source separation algorithm. En *Blind speech separation*. Springer, Dordrecht, 2007. p. 217-241.

[7] RAFII, Zafar; PARDO, Bryan. Repeating pattern extraction technique (REPET): A simple method for music/voice separation. *IEEE transactions on audio, speech, and language processing*, 2012, vol. 21, no 1, p. 73-84.

[8] DURRIEU, Jean-Louis; DAVID, Bertrand; RICHARD, Gaël. A musically motivated mid-level representation for pitch estimation and musical audio source separation. *IEEE Journal of Selected Topics in Signal Processing*, 2011, vol. 5, no 6, p. 1180-1191.

[9] STÖTER et al., (2019). Open-Unmix - A Reference Implementation for Music Source Separation. Journal of Open Source Software, 4(41), 1667, https://doi.org/10.21105/joss.01667